Modeling And Control Of A Robot Manipulator

Ndivhuwo Makondo

Department of Electrical Engineering

University of Cape Town

A dissertation submitted for the degree of

Master of Science in Engineering

Pretoria, March 2013

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author . . . . . .

Pretoria 15 March 2013

Acknowledgements

And I would like to acknowledge my university supervisor, Prof. Martin Braae, for his constant guidance and advice throughout this work.

I would also like to thank my CSIR supervisors, Dr. Nkgatho Tlale and Jonathan Claassens, for their insightful discussions and help from the beginning of this project. Many thanks go to CSIR MIAS for sponsoring my Masters project and for allowing me to use their equipments. I would also like to acknowledge Chioniso Dube for allowing me to use her Force Angle Stability Measure source code in my work. Thanks to everyone at MIAS who supported me during my studies including my fellow students. I am also grateful to my house mate and friend Terence Ratshidaho for his support and to Belinda Matebese for her help and insightful discussions regarding the path planning methods.

I am thankful to my loving parents and siblings, Thembi and Pfarelo, for their love, support, encouragement and comfort.

Finally, I would like to thank my girlfriend Nomazwi Ngcungama for her support.

Abstract

This thesis presents work completed on the design of the modeling and path planning components for a robot manipulator mounted on a mobile platform. This platform is for use in the mining safety inspections of the mine roof, i.e., the hanging wall. Currently this process is done by mine workers and it places them at risk of falling of unstable rocks from the roof. A geometric based inverse kinematics algorithm for a 5 DOF redundant manipulator is proposed and implemented on a Packbot510i used as a test platform. Three versions of the Rapidly-exploring Random Trees planning algorithm namely, basic RRT, RRT Ball and RRT* are compared. Results obtained show that RRT* is more suitable than RRT and RRT Ball in terms of the length and the consistency of the trajectories produced. A Force Angle stability measure is used to guide the robot arm into trajectories that prevent the robotic system from tipping over. Results show that the Force Angle stable measure is more cautious, i.e., it classifies trajectories close to the instability of the system as unstable. Simulation results provided show that this system is capable of carrying out the safety inspections of the roof in the mining environment. Experimental results show that a few modifications are required for the system to be used practically on the test platform due to issues experienced with the hardware.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

2D Two Dimensional

3D Three Dimensional

5D Five Dimensional

CSIR Council for Scientific and Industrial Research

MIAS Mobile Intelligent Autonomous Systems

ESD Electronic Sounding Device

DOF Degrees Of Freedom

RRT Rapidly-exploring Random Trees

RRT* Rapidly-exploring Random Trees Star

RRG Randomly-exploring Random Graph

JT-RRT Jacobian Transposed directed Rapidly-exploring Random Trees

GPS Global Positioning System

IK Inverse Kinematics

DH Denavit-Hartenberg

WAM Whole Arm Manipulator

RPP Randomized Path Planner

ESP Expansive Space Trees

SBL Single-query Bidirectional Lazy

GA Genetic Algorithm

PRM Probabilistic Road Map

COG Center Of Gravity

SP Supporting Polygon

ZMP Zero Moment Point

MHS Moment Height Stability

FA Force Angle

RLMA Ryerson Linkage Mechanism Actuator

CPU Central Processing Unit

RAM Random Access Memory

ROS Robot Operating System

Kinematics Variables

θ_1	Turret angle	[rad]
θ_2	Shoulder angle	[rad]
θ_3	Elbow1 angle	[rad]
θ_4	Elbow2 angle	[rad]
θ_5	Tilt angle	[rad]
A_i	Homogeneous Transformation Matrix	[ ]
${T_j}^i$	Transformation Matrix	[ ]
l_i	Manipulator Linkage Length	[meters]
d_i	Manipulator Linkage offset	[meters]

Path Planning Variables

C Configuration Space

C_{free} Configuration Space free of obstacles

C_{obs} Configuration occupied by obstacles

qI, q_{init} Initial Configuration

qG, q_{goal} Goal Configuration

q_{init} Initial Configuration

q_{near} Nearest Neigbhor

q_{rand} Random Configuration

q_{new}	New Configuration
q_{old}	Old Configuration
T	Random Tree

Force Angle Stability Variables

$\mathbf{f_r}$	Force	[newton]
I_i	Normal	[ ]
$\mathbf{d_i}$	Distance from tip-over axis i	[meters]
β	Stability Index	[ ]
θ_i	Tip-over angle measured from tip-over axis i	[rad]
p_i	Position of ground contact point i	[meters]
p_c	Location of center of mass	[meters]
p_{mass_j}	Position of center of mass of object j	[meters]
$\mathbf{d_i}$	Center of mass of object j	[meters]
m_j	Mass of object j	[grams]
m_{tot}	Total mass of the system	[grams]
a_i	Tip-over mode of point i	[meters]
f_i^*	Effective net force vector	

Chapter 1

Introduction

1.1 Background

To date, robot manipulators have been very successful at manipulation in simulation and controlled environments such as factories. Outside of controlled environments, they have only performed sophisticated manipulation tasks when operated by a human. They have become an integral part in almost all modern manufacturing processes, performing tasks that are considered too dull, repetitive, and hazardous for humans, or that require strength, skill, and precision beyond the capability of humans [1]. There are tasks that are far more dangerous for humans to perform in the outside world, for example, accidents occur in mines during the blast session. One solution to this problem is to investigate and design robotic systems that can perform tasks in a mining environment. An example of these systems could be a mobile manipulator, which is a mobile robot with a manipulator mounted on it for manipulation tasks such as probing the mine roof with an electronic sounding device (ESD) for safety inspection of the roof [2]. Various techniques of designing manipulator systems that work in uncontrolled environments are being developed and they need to be extended to applications in the mining environments. These techniques include kinematic modeling, tip-over stability prevention and path planning of robotic manipulators. This background is based on an extensive survey of the literature that will be dealt in Chapter 2.

Kinematic modeling of a robot manipulator is concerned with the description

of a manipulator's motion with respect to a fixed reference frame in Cartesian coordinate form ignoring the forces and moments that cause motion of the structure. The availability of a kinematic model of robot arm allows us to control the arm and derive path planning algorithms. It is divided into two parts, forward kinematics and inverse kinematics. The forward kinematics determines the position of the end-effector relative to the base frame given the links lengths and the joint angles. The most commonly used method is the Denavit-Hartenberg convention. The inverse kinematics is a process of calculating a set of joint angles which will result in the given end-effector position and orientation. There are generally two methods of solving the inverse kinematics which are closed form methods and numerical methods. Numerical methods are not robot dependent, so they can be applied to any kinematic structure but they iteratively try to find a solution which is time consuming and in some cases they do not compute all possible or acceptable solutions. Closed-form solutions are desirable because they are generally faster than numerical solutions and readily identify a range of all possible solutions. Their disadvantage is that they are not general but robot dependent.

Mobile manipulators operating in a mine environment might face challenges of uneven/rough terrain, slippery ground floor, no GPS or any wireless signal and others. Operation of mobile manipulators on uneven terrain is a requirement for agricultural and field robotics applications, such as search and rescue and mining robotics, where tip-over is a major concern [3]. Robot tip-over occurs when a robot topples over due to an incline or when a manipulator stretches too far out of the base making the system fall. A majority of mobile vehicles are concerned with avoiding tip-over and there are many reasons for robot tip-over prevention. It often results in immobilizing the robot until it can be put back to position by a human or another machine, which may never occur. Tip-over can also create dangerous situations for operators and bystanders, and it can cause collateral damage to humans, other robots, or the general surrounding environment. Additionally, tip-over can delay production and reduce productivity of the mission.

The safety and productivity of these mobile manipulators can be improved by automatic detection and prevention of tip-over instabilities. In order to ac-

complish this, an appropriate measure of the tip-over stability margin must be defined. While a robot may be able to self-right using its manipulator, these strategies are not necessary if the robot can avoid tipping over [4] [5]. This tip-over stability margin is used in the path planning problem to generate motions that do not result in the robot tipping over.

The goal of a typical path planning algorithm is to compute a collision-free path between two given placements of a given robot in an environment populated with obstacles. Obstacles can be physical objects in the environment detected using sensors or regions that the robot is not allowed to go or pass through such as regions resulting in a robot tip-over. There are generally two classes of methods of solving the path planning problem which are combinatorial or exact planning and sampling-based methods. Combinatorial planning solutions construct a discrete representation of the problem that exactly captures the solution which means there are no approximations or sampling errors. These methods are called complete because for every problem they find a solution if it exists; however, they are rarely implemented due to numerical issues and inefficiency due to combinatorial explosion. Sampling-based approaches are by far the most common choice for industrial-grade problems. These practical planners satisfy a weaker form of completeness, i.e, they use randomization to treat the high dimensionality of the state space. The solution produced by these planners might be jagged and making the robot follow such path is problematic. Fortunately, it is straightforward to produce a cleaner path once a jagged solution is given by means of path smoothing.

1.2 Problem Statement

The problem addressed in this thesis is the design and implementation of a path planning system for a mobile manipulator to perform safety inspections in the South African Gold mines. This manipulator is part of a platform used for testing algorithms developed for the CSIR's (Council for Scientific and Industrial Research) mine safety platform project. The aim of the project is to develop a mining platform that can enter stope environments of gold mines directly after blasting to perform safety inspections. Mining of gold in underground mines is

carried out in stopes which are about 1 meter in height, 30 meters in length and 30 meters in breadth. The safety inspections are performed by probing the hanging wall (ceiling) to check if there are any loose rocks. This mining platform consists of a robot manipulator mounted on a mobile base that will have an electronic sounding device (ESD) mounted on its end-effector. The robot manipulator's function is to position the ESD close to the hanging wall for the probing process.

This requires that the robot kinematics are understood and the kinematic model is derived, tip-over stability assessment is done and path planning algorithms are used to plan a path that a robot will follow without tipping over. The iRobot Packbot510i robot is used as a test platform for this study. The main focus of the study is to enable a robot manipulator to execute motions that do not result in the system tipping over so collision avoidance falls beyond the scope of this work. Collision detection algorithms can easily be incorporated in this work to build a collision avoidance system. The data sheet of the test platform is not provided by the supplier so all the dimensions and masses of the robot are approximated and they affect the performance of the system. It is assumed that the map of the environment already exists and only the position of the end-effector goal in Cartesian frame is needed by the system. This work does not focus on the low level control of the individual joints of the manipulator, it should be noted that there could be small errors in the manipulator's input and output joint commands.

The goals of the presented study are as follows:

- To investigate techniques of modeling robot manipulators.
- To derive the kinematic model of the given iRobot Packbot510i manipulator.
- To investigate various tip-over stability measure algorithms and implement the most suitable technique for the project.
- To investigate various motion or path planning algorithms and implement the most suitable technique for the project.
- To implement these algorithms on a Packbot510i manipulator, shown in

Figure 1.2, in order to perform mine safety inspections on any South African gold mine.

Figure 1.1: The iRobot Packbot Robot used as the platform for the Mine Safety Platform Project at the Mobile Intelligence Autonomous Systems Lab at the CSIR.

1.3 Methodology

A forward kinematics model of the robot manipulator is derived using the well known DH convention. A geometric technique is used to solve the inverse kinematics problem of the robot manipulator. After analysis of inverse kinematics literature and background work, a geometric based inverse kinematics was found suitable for this project due to the speed of execution and its ability to readily identify a range of all possible solutions. A literature study on robot tip-over instability was done and the Force Angle stability measure was chosen. Fortunately a Force Angle model of the iRobot Packbot manipulator has been developed by Dube in [6]. Source code of this Force Angle stability measure model of the Packbot manipulator was readily available and is implemented here.

Various path planning algorithms exist in the literature, sampling-based algorithms were chosen because they are practical. Three algorithms are implemented and tested through simulation experiments, before being used in the final system. These algorithms fall under the Rapidly-exploring Random Trees family. The

first algorithm is the basic RRT and the rest are the optimal versions called RRT Ball and RRT*. Firstly, these three algorithms are tested without considering the stability of the system and the environmental constraints (height restriction imposed by the mine stope). This is done to test the computational intensity and the reliability of each algorithm. Algorithms that pass the first test are then tested considering the stability of the system. This is done by using the Force Angle stability criterion as a "collision detection" module in the planning stage. The arm configurations that result in the instability of the system are considered to be in a collision space and therefore are rejected. Finally, the algorithm that performs best in the second test is used to operate the manipulator under a height restriction of 1 meter and above the ground without tipping over the system. So the final system uses two sets of collision detection modules, one checks whether any part of the arm will hit the ground or go over the roof and the other checks if the system is stable or not. Given the desired (x, y, z) goal position, the full system should be able to identify the corresponding joint angles using the kinematic model, plan a path to the desired goal using a chosen RRT planner and ensuring that the motions executed by the robot are stable by checking with the Force Angle stability measure.

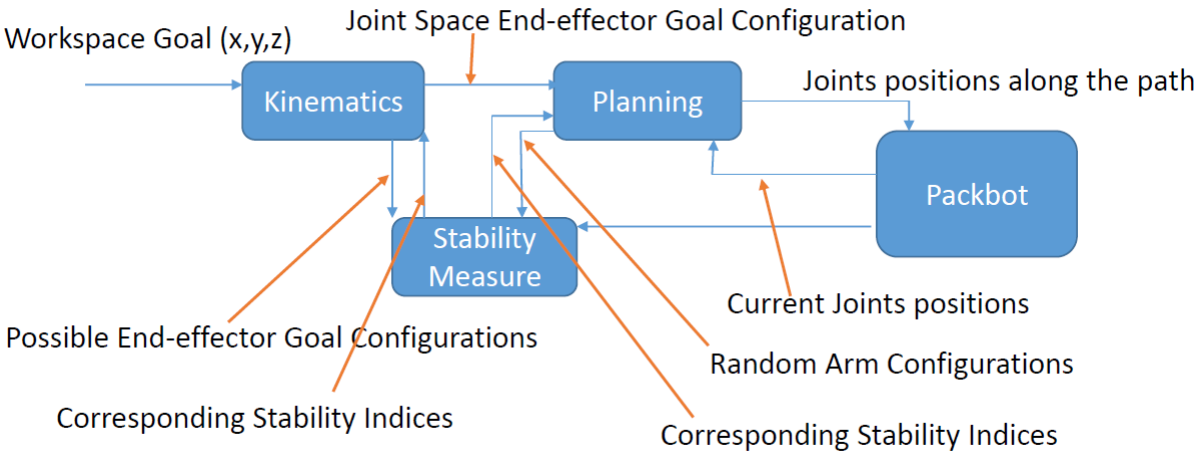


Figure 1.2: Flowchart showing three stages of the proposed method and how the Packbot is controlled.

1.4 Thesis Outline

This thesis discusses the use of a robot manipulator to perform mine safety inspections. A literature study on the components needed in this thesis is presented in Chapter 2. Kinematics modeling of the iRobot Packbot510i manipulator is done in Chapter 3. In Chapter 4, the path planning algorithms are designed for the test platform. Chapter 5 shows how the stability index is computed for the system stability assessment. Simulation results on each component are presented in Chapter 6 and experimental results of the combined system verifying and validating the various system components are presented in Chapter 7. Finally, Chapter 8 provides conclusions and recommendation for future approaches, based on the findings of this study.

Chapter 2

Review of Related Work

In this chapter, literature related to kinematics modeling, path planning and tip-over stability measure for robot manipulators is discussed. As previously mentioned, the goal of this work is to use a robot manipulator mounted on a mobile base to perform safety inspections in the South African gold mines. This requires that kinematics of the manipulator are understood and modeled and path planning algorithms are developed to guide the manipulator through a path towards the goal. Robot manipulators mounted on a mobile base face a danger of tipping over the system during operation, this requires that some sort of stability measure is used to make sure that the path that the planner produces results in a stable motion of the system.

Initially, kinematics modeling of robot manipulators is investigated in order to find a suitable method of modeling the robot under study, followed by a review of various manipulator path planning algorithms. Finally, literature on the computation of a tip-over stability measure is presented.

2.1 Kinematics Modeling

In this section, methods used for modeling manipulator kinematics are investigated in order to find a suitable method that can be used to model the robot under study. Basic knowledge of manipulator kinematics is assumed in this section otherwise the reader is referred to Appendix A for an introduction. Kine-

matic modeling is one of the most essential analytical tools of robotics. It is used for modeling mechanisms, actuators, and sensors for on-line control, off-line programming, and simulation purposes [7]. Kinematics analysis of a robot manipulator is concerned with the description of the manipulator's motion with respect to a fixed reference frame in Cartesian coordinate form ignoring the forces and moments that cause motion of the structure. It describes the analytical relationship between joint positions and the end-effector position and orientation. The availability of a manipulator's kinematic model allows us to derive path planning algorithms for the manipulator. Manipulator kinematics modeling is comprised of two parts, forward kinematics and inverse kinematics.

2.1.1 Forward Kinematics

The forward kinematics problem for a serial chain manipulator is to find the position and orientation of the end-effector relative to the base given the positions of all the joints and the values of all the geometric link parameters. It concerns the determination of a systematic, general method to describe the end-effector motion as a function of the joint motion by means of linear algebra tools [8]. In practice, the forward kinematics problem is solved by calculating the transformation between a reference frame fixed in the end-effector and another one fixed in the base. The objective of forward kinematics is to determine the cumulative effect of the entire set of joint variables in the serial chain. Various conventions that provide a systematic procedure for performing this analysis have been developed in the literature. It is possible to carry forward kinematics on simple mechanisms without respecting these conventions. However, the kinematic analysis of an n-link manipulator can be extremely complex and the conventions introduced simplify the analysis considerably. Moreover, they give rise to a universal language with which robot engineers can communicate [9].

A commonly used convention for selecting frames of reference in robotic applications is the Denavit-Hartenberg, or D-H convention which is explained in Chapter 3. In this convention, each homogeneous transformation A_i is represented as a product of four basic transformations: the rotation about the z-axis, the translation along the z-axis, the translation along the x-axis, and the rotation

about the x-axis.

A robot with n joints will have $n+1$ links, since each joint connects two links. Joints are numbered from 1 to n, and links from 0 to n, starting from the base. By this convention, joint i connects link $i-1$ to link i. The location of joint i is considered fixed with respect to link $i-1$. With the ith joint, a joint variable is associated, denoted by q_i. In the case of a revolute joint, q_i is the angle of rotation and in the case of a prismatic joint, q_i is the joint displacement:

$$q_i = \theta_i \tag{2.1}$$

for joint i revolute and

$$q_i = d_i \tag{2.2}$$

for joint i prismatic

To perform the kinematic analysis, a coordinate frame is rigidly attached to each link. Now suppose A_i is the homogeneous transformation matrix that expresses the position and orientation of $o_i x_i y_i z_i$ with respect to $o_{i-1} x_{i-1} y_{i-1} z_{i-1}$. The matrix A_i is not constant, but varies as the configuration of the robot is changed. However, the assumption that all joints are either revolute or prismatic means that A_i is function of only a single joint variable, namely q_i. In other words,

$$A_i = A_i(q_i) \tag{2.3}$$

Now the homogeneous transformation matrix that expresses the position and orientation of $o_j x_j y_j z_j$ with respect to $o_i x_i y_i z_i$ is called, by convention, a transformation matrix, and is denoted by T_j^i. if $i < j$

$$T_j^i = A_{i+1} A_{i+2} ... A_{j-1} A_j \tag{2.4}$$

if $i = j$

$$T_j^i = I \tag{2.5}$$

if $j > i$

$$T_j^i = (T_i^j)^{-1} \tag{2.6}$$

It is important to note that there is a different set of DH parameters that

is also used, due to Craig [10]. The big difference in Craig's convention is that coordinate origin i for link i is located at the nearest joint. An advantage of Craig's convention is the proximal placement of the origin for a link. Also the rotation θ_i is about z_i and the joint number is the same as the coordinate number, which seem more natural. Torque exerted about joint i is also at the same place as at the coordinate system of link i, to which inertial parameters such as center of mass are likely to be referenced. A disadvantage is that the transform mixes $i-1$ and i parameters. The interested reader is referred to [10] for more information about this method. The conventional DH parameters are used to model the robot in this study since it has become widely used by robotics engineers. The application of this convention to the Packbot manipulator used in this study is shown in Chapter 3.

2.1.2 Inverse Kinematics

This section deals with the inverse kinematics (IK) modeling of redundant serial manipulators such as the manipulator under study. A robot manipulator with more degrees-of-freedom (DoF) than are required to perform a given task is called a kinematically redundant manipulator. The redundant DoF provides the task execution with flexibility and adaptability [11]. Redundant manipulators have multiple or infinite solutions of the joint angles for a given end-effector task. The choice of the one suitable solution among all possible solutions to achieve a second subtask is the main concern in studying such manipulators [12]. A performance criterion is generally introduced to resolve the redundant DoF in the solutions.

Singularity avoidance is one of the ways of utilizing redundancy as a second subtask [13]. A manipulator is said to be at singular position when it loses one or more degrees of freedom, and it will not be able to move in some directions in the end-effector space. There is another type of singularity called algorithmic singularity which arise when the constraint task conflicts with the end-effector task. To overcome this type of singularities, either the constraint function must be keenly specified case-by-case or singularity robust techniques must be adopted to invert the augmented or the extended Jacobian matrix [14]. Many researchers have used the manipulability measure as a way of utilizing the redundancy of

robot arms [15]. The manipulability measure gives an indication of how 'far' the manipulator is from singularities.

Many authors have used the concept of stability measure for mobile manipulators as another way to solve the redundancy problem [16]. In order for a mobile manipulator to move stably (not overturn) and execute the given motions of the end-effector and the vehicle simultaneously, a manipulator must have redundancy. By using this redundancy, it is possible to perform tasks at optimal manipulation configuration when the robot is stable, recovering the system's stability when the robot is unstable [16].

Collision avoidance can also be used as a metric for redundancy resolution for robot manipulators [17]. When the arm robot control system detects a collision, it modifies the arm joints trajectories in order to increase the distance between the robot and the obstacle and, at the same time, maintain the initial trajectory of the end effector. Lastly, redundancy has also been widely leveraged for solving the joint limit avoidance problem [18]. The joint limit is the range which the joint is allowed to move. If these limits are exceeded, it may lead to nominal commands in the joint space exceeding some bounds, with an associated saturation that makes the resulting robot motion unpredictable [19]. In this study, a stability measure is leveraged as a way of utilizing the redundancy of the manipulator under study.

Before any robot manipulator can be controlled to do any task, we must have a way to determine its inverse kinematics solution. Inverse kinematics refers to the process of determining an allowable set of joint angles which will result in a specific position and orientation of the end-effector [20]. This problem can be trivial for non-redundant manipulators since if the solution exists only one solution exits.

There are generally two methods of solving the inverse kinematics problem which are closed-form methods and numerical methods. Numerical methods are not robot dependent, so they can be applied to any kinematic structure. Their disadvantages are that being iterative they can be slower and in some cases, they do not compute all possible or acceptable solutions and often suffer from singularities. A numerical method typically will continue to make adjustments to approach a local minima, even if a more optimal solution exists. Numerical

methods include cyclic co-ordinate descent [21] [22] [23] , Jacobian based, neural networks [24] [25] [26], inverse-forward scheme [27] , offset modification [28], damped least-squares, quadratic programming and other methods.

Closed-form solutions include geometric methods [12] [18] [29] [30] [31] [32] and algebraic methods [33] [34] [35] [36] . Closed-form solutions are desirable because they are generally faster than numerical solutions and readily identify a range of all possible solutions. The disadvantage of closed-form solutions is that they are not general, but robot dependent. In addition, these solutions are not only manipulator dependent but are also subject to uncertainty due to manufacturing errors. Geometric methods involve decomposing the problem into separate planar problems and solving each problem using algebraic manipulation. Algebraic methods involve identifying the significant equations containing the joint variables and manipulating them into a soluble form. The algebraic approach suffers from the fact that the solution does not give a clear indication of how to select the correct solution from the several possible solutions for a particular arm configuration [12]. Authors use various strategies to the challenging inverse kinematics using the algebraic approach [33].

Mohamed et al in [12] developed a geometrical method to solve the inverse kinematic problem for hyper redundant manipulators. Their method finds one solution from many solutions by making sure that the angles between each adjacent links are the same which is good for avoiding singularities. Yahya et al in [32] developed a motion planning algorithm using the inverse kinematics developed in [12] for a robot manipulator with equal links. Singh and Claassens developed a geometric solution for the 7 DOF Barrett Whole Arm Manipulator with link offsets in [31]. They showed that all possible geometric poses can be completely defined by three circles in the Cartesian space. Lee in [29] solved the inverse kinematics for PUMA robots and found out that they have eight solutions resulting from different arm, elbow and wrist configurations. The PUMA robots do not fall under redundant manipulators that have infinite or many solutions.

In [30], another geometric method for hyper-redundant manipulators is proposed. The authors there try to solve the inverse kinematics for a planar manipulator with n-links serially connected together. Their aim is to minimize the change in each joint angle. All methods described above do not seem to find the

range of feasible solutions. The method in [31] finds feasible solutions for the Barrett WAM. Moradi and Lee [18] extended the method presented by Dahm and Joublin for a 7 DOF arm and worked the limits for the redundancy angle parameter for the shoulder joint limits. The idea used in this study is similar to this in a sense that it finds the limits of the shoulder joint (θ_2 in this case).

In the case of the iRobot Packbot used in this study, there is an interval of configurations which yield a specific end-effector pose. This arises from the redundancy of one of the joints. If the range of possible configurations that yield a specific pose could be analytically determined, an 'optimal' configuration that better satisfies some metric such as stability can be chosen. Numeric IK schemes would simply find a specific solution. A geometrical method is used in this study and it is explained in Chapter 3.

2.2 Path Planning

A typical motion planning problem requires the computing of a collision-free motion between two given placements of a given robot in an environment populated with obstacles [37]. Most applications require that the path found be optimal according to some metric. The problem is typically solved in *the configuration space C*, in which each placement (or configuration) of the robot is mapped as a point [38]. A complete specification of the location of every point on the robot is referred to as a configuration, and the set of all possible configurations is referred to as the configuration space. As for a two link arm, a vector of two joint angles represents this configuration. The free configuration space C_{free} is the subset of the configuration space at which the robot does not intersect any obstacles. The robot can move from an initial to a goal configuration without intersecting an obstacle if the two configurations lie in the same connected component of C_{free}. The figure below shows a configuration space with the starting configuration qI, goal configuration qG, obstacle free configurations C_{free} and configurations occupied by obstacles C_{obs}. The goal of the motion planner is to find the connection between qI and qG as shown by the curve in Figure 2.1.

There are generally two classes of methods of solving the path planning problem. One class of planning methods is combinatorial or exact planning. Com-

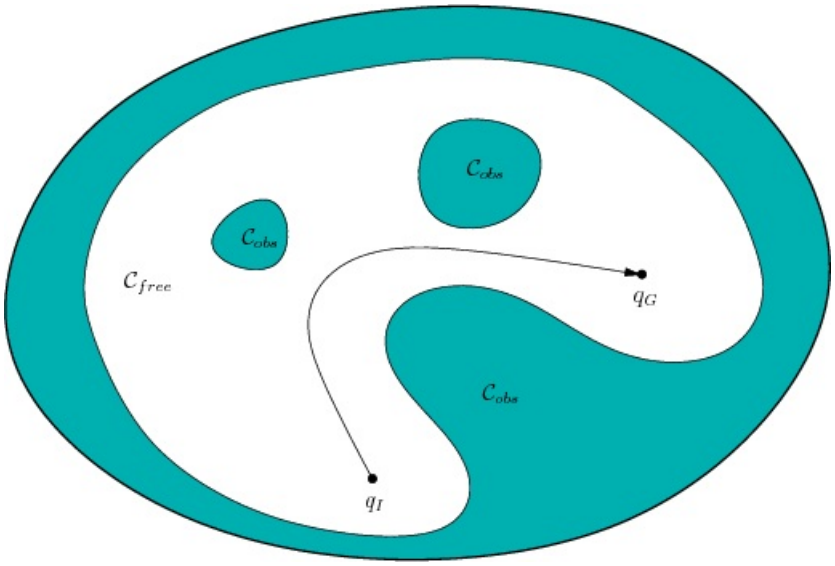


Figure 2.1: The configuration space [39]

binatorial planning solutions construct a discrete representation of the problem that exactly captures the solution. In other words, there are no approximation or sampling errors. These methods are called complete, meaning that, for any input problem, they correctly determine in finite time whether or not a solution exists. Most motion-planning problems involve robots that are not modeled as points and they can rotate in addition to translating. Combinatorial planning methods experience difficulty extending to those situations. Constructing C_{obs} in terms of polynomial roots is straightforward, but a combinatorial explosion occurs that produces far too many facets to be practical. For 3D problems, it becomes considerably worse. The next difficulty is to perform cell decomposition. Methods exist that provide solutions to the general path planning problem; however, they are rarely implemented due to numerical issues and inefficiency due to combinatorial explosion [39]. This class of methods will not be considered in this study.

Another class of planning methods is sampling-based methods. Sampling-based approaches are by far the most common choice for industrial-grade problems, because the configuration space in which obstacles lie (C_{obs}) is composed of an unwieldy number of facets. These practical planners satisfy a weaker form

of completeness, i.e., they use randomization to treat the high dimensionality of C. The term *probabilistically complete* was introduced to characterize these sampling-based algorithms, able to find a solution if sufficient running time is given. A common nuisance with sampling-based is that the produced paths are jagged as they traverse C_{free}. This makes the solution animation jumpy; making the robot follow such awkward paths is problematic. Therefore, path smoothing is usually performed to clean up solution paths. Fortunately, it is straightforward to produce a cleaner path once a jagged solution is given [37] [39]. Examples include the randomized path planner (RPP) [40], Aridne's clew [41], Expansive Space Trees (EST), Single-query Bidirectional Lazy road map planner (SBL), Genetic Algorithms (GA), probabilistic road map planners (PRM) [42], and rapidly exploring random trees (RRT) [43]. Each of these methods can be seen as belonging to a field called *sampling-based motion planning* [44].

RPP planner combines a gradient descent of the potential with a random walk procedure to escape the local minima. The RPP approach is probabilistically complete and has provided very good results. However, it is now well known that this planner is hindered by narrow passage problems [37], [45]. Ariadne's clew approach grows a search tree that is biased to explore as many new regions as possible in each iteration. There are two modes, SEARCH and EXPLORE, which alternate over successive iterations. The drawback attributed to this approach is that the optimization process carried out by EXPLORE is costly and may require some parameter tuning [37], [45]. The EST planner presented in [46] and [47] shares some ideas with PRM approaches, it tries to sample only the portion of C that is relevant for a particular query, avoiding the cost of precomputing a road map for the whole free-space. The algorithm iteratively executes two steps: *expansion* and *connection*, in a similar way used in ACA. This is a bidirectional planner (i.e it constructs two trees), although a unidirectional version is also implemented. The EST uses a weighting function to avoid oversampling in regions already explored and rather to bias the expansion towards unexplored areas of the C. In this respect both RRT and the expansive planner aim for the same goal, the only difference being in the technique used to identify poorly explored zones [37]. The main drawbacks are that the planner requires substantial parameter tuning, which is problem-specific (or at least specific to a similar family of problems), and

the performance tends to degrade if the query requires systematically searching a long labyrinth path. Choosing the radius of the predetermined neighborhoods essentially amounts to determining the appropriate resolution.

SBL planner is another approach issued from the probabilistic road map framework for single planning queries [48]. In this case, a roadmap is not built trying to cover the whole C. The idea is to exploit the delayed collision checking by combining it with an adaptive sampling technique similar to the one used by EST. The planner searches C by building a road map made of two nodes, T_s and T_g. The root of T_s is the start configuration and the root T_g is the goal configuration (i.e it uses bidirectional search). Every new node generated during the planning is installed in either one of the two trees as a child of an already existing node. The link between the two nodes is the straight-line segment joining them in C. This segment will be tested for collision only when it becomes necessary to perform this test to prove that a candidate path is collision-free (lazy collision checking). This algorithm has been applied to problems involving several manipulator arms operating in the same workspace [37] [49].

Genetic Algorithms are search procedures based on the mechanics of genetics and natural selection. They combine an artificial survival of the fittest with genetic operators abstracted from nature to form a surprisingly robust search mechanism that is suitable for a variety of search problems. The generation-by-generation evolution of the population of potential solutions results in a convergence towards the best possible solution. Genetic algorithms may sometimes have difficulties to converge in an optimum solution. This is mainly due to some phenomena such as deceptive problem and the problem of premature convergence. GA has some drawbacks in terms of consistency of the solution, time consumed and parameter tuning [37], [50].

The probabilistic road map planner is a relatively new approach to motion planning and is one of the leading motion planning techniques. It was developed simultaneously at Stanford and Utrecht [42]. It turns out to be very efficient, easy to implement, and applicable for many different types of motion planning problems. Globally speaking, the PRM approach samples the configuration space for collision-free placements. These are added as nodes to a road map graph. Pairs of promising nodes are chosen in the graph and a simple local motion planner is

used to try to connect such placements with a path. This process continues until the graph covers the connectedness of the space [37], [45]. Although PRMs can often find a solution to solving the path finding problem the solutions are often not practical in that they can cause the device to fail around obstacles or to pass very close to them in the environment [51].

The PRM algorithm and its variants are multiple-query methods. In some applications, computing a road map a priori may be computationally challenging or even infeasible and not suitable for on line path planning. Incremental sampling-based planning algorithms such as RRTs have been developed as on line counterparts to the PRMS. The incremental nature of these algorithms allows termination as soon as a solution is found as opposed to constructing a graph of the entire environment. This reduces the computational burden of the planning algorithm [52]. A single query planner is suitable for this study as the manipulator tries to go from the start to a single goal configuration which means PRMs are not suitable for this study.

Rapidly exploring Random Trees-based (RRT) methods fall into a larger family of methods called incremental sampling and searching, in which a graph is incrementally constructed inside C_{free}. During the last decade, incremental sampling-based motion planning algorithms have been shown to work well in practice and to possess theoretical guarantees such as probabilistic completeness without requiring parameter tuning [52]. The term probabilistic completeness means the probability that the algorithm will find a solution if it exists approaches 1 as planning time increases and will return false if no solution exists. The original RRT algorithm is known to provide feasible solutions which are not always optimal for a given cost function. The basic RRT construction is explained in Chapter 4.

The RRT algorithm has some useful properties that are listed in [39] as: 1) the expansion of an RRT is heavily biased toward unexplored portions of the state space; 2) the distribution of vertices in an RRT approaches the sampling distribution, leading to consistent behavior; 3) an RRT is probabilistically complete since it falls under sampling-based algorithms; 4) the RRT algorithm is relatively simple, which facilitates performance analysis; 5) an RRT always remains connected, even though the number of edges is minimal; 6) an RRT can be considered as a

path planning module, which can be adapted and incorporated into a wide variety of planning systems; 7) entire path planning algorithms can be constructed without requiring the ability to steer the system between two prescribed states, which greatly broadens the applicability of RRTs. Although RRTs have been implemented successfully since the time of their conception, many challenging issues remained [39]. Research has been done facing these challenges and improvements have been made.

Using the basic RRT algorithm without any bias toward the goal causes the algorithm to converge slowly to the goal. An improved planner, called RRT-GoalBias, can be obtained by replacing RANDOM_CONFIG in Algorithm 2 with a function that tosses a biased coin to determine what should be returned. If the coin toss yields "heads", then x_{goal} is returned; otherwise, a random state is returned. If too much biasing to the goal is introduced, the planner can be trapped in a local minima. See [54] for more goal biasing RRT algorithms. RRT-Connect planner is designed specifically for path planning problems that involve no differential constraints [43]. This method is based on two ideas: the Connect heuristic that attempts to move over a longer distance, and the growth of RRTs from both start and goal configurations. Instead of attempting to extend an RRT by a single ϵ, the Connect heuristic iterates the *EXTEND* step until q or an obstacle is reached. Two trees, T_a and T_b are maintained at all times until they become connected a solution is found.

Lindermann and Lavalle in [55] discuss theoretical and practical issues related to using RRTs to incrementally reduce dispersion in the configuration space. They introduce RRT-like planners based on exact Voronoi digram computation, as well as sampling-based which approximate their behavior. They discuss how to use sampling techniques to increase the RRT's voronoi bias, with the goal of searching in a way that incrementally reduces dispersion. Dispersion is a measure of how well a space is covered by a sample set. They deduce that focusing on dispersion reduction promotes growth toward unexplored regions and uniform coverage of the configuration space. They give experimental results showing how the new algorithms explore the configuration space and how they compare with existing RRT algorithms. Initial results show that their algorithms are advantageous compared to existing RRTs, especially with respect to the number of collision

checks and nodes in the search tree [55].

Authors of [56] developed a novel optimization-based retraction algorithm to improve the performance of sampling-based planners in narrow passages for 3D rigid robots. The retraction step is formulated as an optimization problem using an appropriate distance metric in the configuration space. Their algorithm computes samples near the boundary of C-obstacles using local contact analysis and uses those samples to improve the performance of RRT planners in narrow passages. Ferguson and Stentz in [57] presented an anytime algorithm for planning paths through high-dimensional, non-uniform cost search spaces. Their approach works by generating a series of Rapidly-exploring Random Trees where each tree reuses information from previous trees to improve its growth and the quality of its resulting path. They also presented a number of modifications to the RRT algorithm that they use to bias the search in favor of less costly solutions. The resulting approach is able to produce an initial solution very quickly, then improve the quality of this solution while deliberation time allows. It is also able to guarantee that subsequent solutions will be better than all previous ones by a user-defined improvement bound [57].

In [52] Karaman and Frazzoli developed two extensions of the RRT called Rapidly-exploring Random Graph (RRG) and RRT*. It is proven in [52] that under technical conditions, the cost of the best path returned by RRT converges almost surely to a non-optimal value, as the number of samples increases. It is shown that the cost of the best path returned by RRG converges to the optimum almost surely. The RRT* preserves the asymptotic optimality of RRG while maintaining a tree structure like RRT. In terms of computational complexity, it is shown that the number of simple operations required by both the RRG and RRT* algorithms is asymptotically within a constant factor of that required by RRT.

Akgun and Stilman in [58] present a sampling-based motion planner that improves the performance of the probabilistically optimal RRT* planning algorithm developed by Karaman and Frazzoli in [52]. Their experiments demonstrate that their planner finds a fast initial path and decreases the cost of this path iteratively. They identify and address the limitations of RRT* in high dimensional configuration spaces. They introduce a sampling bias to facilitate and acceler-

ate cost decrease in these spaces and a simple node-rejection criteria to increase efficiency. Finally, they incorporate an existing bidirectional approach to search which decreases the time to find an initial path. They analyze their planner on a simple 2D navigation problem in detail to show its effectiveness. Their results consistently demonstrate improved performance over RRT*.

A similar approach is introduced by Krammer et.al. in [59] for robot motion planning especially for car-like robots. The presented approach uses a pre-computed auxiliary path to improve the distribution of random states. The main contribution is the significant increased quality of the computed path. A proof-of-concept implementation evaluates the quality and performance of the proposed concept. By using a pre-computed auxiliary path to guide an RRT to a goal configuration they increase the probability of the nodes near the path to be randomly selected for exploration. This is similar to RRT* in the sense that they both improve on a path that already exists. RRT* uses the original RRT algorithm to compute the original path that it later improves whereas the approach by Krammer uses a pre-computed path that could be generated using a different algorithm.

Lacevic et.al in [60] present a sampling-based motion planning approach, for articulated manipulators, that generates safe paths. Their approach uses the RRT paradigm to establish a collision-free path in the configuration space. The expansion of the trees is influenced by a modified version of the kinetostatic danger field - a safety assessment function. The idea is to to grow the trees towards safer regions. Thus, the planner provides not only collision-free paths, but strives for safer ones. This is achieved by embedding a danger field based safety assessment into a heuristic function that dictates the expansion of trees. In this case a safe path means a path that is some distance away from the obstacle. The heuristic function can easily be incorporated into some well-known algorithms without altering their structures. They proposed two versions of the planner. The first is the modification of the Jacobian transposed directed RRT(JT-RRT) algorithm that grows a single tree from the start configuration and uses the transpose of the Jacobian to guide the sampling towards the goal defined in the workspace. The second is an extension of the standard bidirectional RRT-connect planner where the inputs to the algorithm are the start and the goal

configuration that serve as seeds for the trees to grow. Their simulations show how the proposed algorithms substantially improve the quality of the obtained paths in terms of safety when compared to existing algorithms. The initial results with the SAFE_RRT_CONNECT algorithm hint that fewer nodes are used compared to the original RRT_CONNECT algorithm [60].

In this study, the basic RRT in [39], the RRT* and another version of the RRT* in [52] referred to as RRT Ball will be implemented on the iRobot Packbot510 manipulator and results will be compared in order to choose a more suitable algorithm. These planners must use a stability measure, discussed in the next section, as a collision detection module to generate stable paths.

2.3 Tip-over Stability Measure

This section deals with the modeling and computation of a stability measure used in tip-over instability prevention. Mobile machines equipped with manipulators operating autonomously are starting to be common systems in the mining and search, and rescue environments. These systems are given critical tasks and sent on dangerous missions. When these systems operate over very uneven or sloped terrain, tip-over instabilities may occur which risks damaging the system, and a critical mission to be aborted. In the worst case scenario, this event will place a human in harms way during the robot recovery. This instability can delay production and reduce productivity of the mission. In this study, operation of the manipulator is allowed only when the robot is in a static position, therefore only static stability is considered. If the manipulator was allowed to operate while the robot system was in motion the dynamics of the complete system would need to be considered.

For static stability, the Center of Gravity (COG) must lie above the convex area spanned by the ground contact points [3]. This is called the supporting polygon (SP) principle which according to [61] is arguably the most influential tip-over stability criteria. A low center-of-mass is always desirable from a tip-over stability point of view, heaviness on the other hand is stabilizing at low velocities, and destabilizing at high velocities [5].

There are three commonly used stability measures amongst others in the

literature namely Zero-Moment Point (ZMP), Moment-Height Stability (MHS) and Force Angle (FA) stability measure. The ZMP is defined as the point on the ground about which the sum of all the moments of active force is equal to zero. It was originally derived for stability analysis for bipedal robots, and it has been adapted to other types of mobile robots. As described in [62] if ZMP is inside the support polygon, the mobile manipulator is stable. The reader is referred to [62] for more details about the ZMP criterion.

Moment Height Stability measure was proposed by Moosavian and Alipour in [63] for wheeled mobile manipulators. The proposed metric is physically meaningful based on principal concepts, and can be implemented with limited low computational effort. The suggested MHS can be effectively used for both legged robots and mobile manipulators. See [63] for more information about the MHS measure. The Force Angle (FA) stability measure was first proposed by Rey and Papadopoulos in 1996 [64]. It is easily computed and sensitive to top heaviness. The proposed metric is applicable to systems subject to inertial and external forces, operating over even or uneven terrains.

Moosavian and Alipour investigated various dynamic stability measures and compared each other via simulations. It was shown that FA is too confident while the ZMP is too cautious compared to others. Simulation results based on a wheeled mobile robot show the merits of the MHS measure, in terms of prediction of the exact time of instability occurrence, without additional precautions of the other measures which may confine the maneuverability of the system and its operation [63]. Roan et al. [4] collected data of a mobile robot tipping over and then compared this data to the stability measures provided by three measures in order to verify that these algorithms accurately match real-world behavior. The stability measures used in this comparison are ZMP, FA and MHS. A small mobile robot platform based on the iRobot PackBot drove a course including ramps and obstacles; an IMU and GPS provided inertial and positional data for the algorithms, and the actual tip-over event is determined form video footage of the tests. It was found that the measures show a significant amount of noise, which is likely due to the vibrations caused by movement of the tracks and could be reduced by employing additional filtering during data collection. They concluded that if noise can be significantly reduced, then the preliminary real-world data

suggests that the FA and MHS algorithms are able to assess robot stability and can be used as part of a tip-over avoidance system.

The Force Angle and Moment Height Stability measures seem to perform better in real-world applications compared to the Zero Moment Point for a mobile manipulator. An investigation on the effects of the manipulator and flipper pose on the stability of a tracked mobile manipulator is carried in [6]. The FA stability criterion is used to compute the stability index of the platform. A model of the iRobot PackBot 510i in relation to the flipper state of contact with the ground is developed. The tip-over stability based on the flipper and manipulator pose is then analyzed. It was found that the pose of the manipulator and flippers of tracked mobile manipulators have a large impact on the tip-over stability of the system. In this case the manipulator and flippers can be used to prevent tip-over of the mobile manipulator.

In this study, the Force Angle stability model for the iRobot Packbot510i manipulator developed in [6] will be used. The Force Angle (FA) stability measure was first proposed by Rey and Papadopoulos in 1996 [64]. A planar example is presented in Chapter 5. Rey and Papadopoulos described the two types of tip-over instability which may occur for mobile manipulators operating over uneven terrain: tip-over in the absence of destabilizing inertial forces (static instability), and tip-over in the presence of destabilizing inertia forces (dynamic instability) [65]. Their work makes use of the dynamic Force-Angle stability margin measure presented in [64], and introduces a static version. A simple and effective tip-over prediction method is described, as well as an algorithm for triggering a tip-over prevention response. The performance of the proposed methods based on the FA measure are described and their performances compared for the case of a simulated foresty vehicle executing an unstable manoeuvre [65].

Beck et al.[66] proposed a mechanism capable of enhancing the safety of paths followed by mobile robots which significantly modify their mass distribution while operating in uneven terrains based on the FA stability measure. A proposed cost function is defined around the stability margin to address four key objectives:

- prevention of tip-over and operation within certain safety limits, the key
- equal distribution of the resulting forces on the supporting points

- operating within nominal joint positions
- low energy consumption

  The first objective in the list above is the main focus of the tip-over stability measure section in this thesis.

Simulation results of the proposed optimized motion planner for an iRobot Explorer tracked vehicle are presented. They are also compared with a non-optimized planners to show the validity of the approach. They found that their results demonstrate marked improvements primarily in tip-over prevention, in particular when compared with simple methods that do not account for resulting forces exerted at the robot support points.

A generic methodology to plan increasingly stable paths for mobile platforms traveling over uneven terrain is proposed in [67]. This is accomplished by extending the Fast Marching level-set method propagating interfaces in 3D lattices with an analytical kyno-dynamic metric which embodies robot stability in the given terrain. The stability metric used in this work is the Force-Angle stability measure. This is particularly relevant for reconfigurable platforms which significantly modify their mass distribution through posture adaption, such as robots equipped with manipulators or varying traction arrangements. Results obtained from applying the proposed strategy in a mobile rescue robot operating on simulated and real terrain data illustrate the validity of the proposed strategy [67].

So far we have discussed tip-over instability prevention based on the ZMP, FA and MHS stability measures. Authors have used other methods for computing the stability index and tip-over prevention. Some of these methods are similar to methods discussed above including the supporting polygon principle (SP). Hatano et al. [68] proposed a stability control method with a criterion based on reaction for motion control of a mobile manipulator. The evaluation method based on ZMP criterion, in which the model is regarded as a particle system, was found not satisfactory for manipulators that consists of solid links. In their work, they discussed the issue of transient state during tipping over and showed using simulations of a mobile robot with 1 DOF manipulator that a mobile manipulator can return from unstable transient state to stable state by performing stability compensation motion.

Moshe and Shiller [69] proposed a unified measure of stability of a Rocker Bogie vehicle that accounts for the tendency to slide, tip-over, or lose contact with the ground considering both static equilibrium and dynamic effects. The measure of stability is computed by solving for the range of acceptable velocities and accelerations that satisfy a set of dynamic constraints. The maximum acceptable velocity serves as a dynamic stability measure, whereas the maximum acceptable acceleration at zero velocity serves as a static stability measure. The utility of the static and dynamic stability margins are demonstrated for both two dimensional and longitudinal quasi-3D motion in simulations.

Morales et al. [70] did a study of the Center of Gravity (COG) for the ALACRANE mobile robot, which consists of a hydraulic tracked vehicle with a customized heavy manipulator. The COG was experimentally estimated by taking into account different arm positions. These estimations were used to modify the mass distribution of the robot so that the COG can be appropriately controlled on line by using on-board inclinometers. The arm base rotation is employed to improve static tip-over stability according to the supporting polygon principle. Experimental results on the ALACRANE robot were discussed.

In [71], the authors analyzed tip-over stability and developed tip-over avoidance algorithms for a reconfigurable tracked mobile modular manipulator negotiating slopes, with consideration of track-terrain and vehicle-manipulator interactions. The criteria are derived on the basis of load transfers, and tip-over avoidance algorithm is developed with track reconfiguration or manipulator adjustment so as to balance load distributions. The effectiveness of the developed algorithms are verified through simulations and experiments on the Ryerson linkage mechanism actuator (RLMA). After looking at all these methods, the FA method seems to produce good results and has been used on Packbot robots before. The Force Angle stability model of the Packbot510i manipulator for the computation of the stability is presented in Chapter 5.

Chapter 3

Kinematics Modeling of the Packbot510i Manipulator

This chapter deals with the kinematic modeling of the robot manipulator under study. Section 3.1 provides the description of the robot manipulator, section 3.2 describes the forward kinematics of this robot and section 3.3 shows how the inverse kinematics solution is derived for this particular robot manipulator.

3.1 Platform Description

Figure 3.1: iRobot Packbot Robot

The platform under study is the iRobot Packbot510i manipulator shown in Figure 3.1. Note that this figure has been repeated here for convenience of referencing. This is a manipulator with 8 independent degrees of freedom. It consists of shoulder rotation (referred to turret angle), shoulder pivot, elbow 1 pivot, elbow 2 pivot, gripper rotation, gripper open and close, head rotation/pan and head tilt. The number of links in this manipulator is 3. The joints required to position the end-effector in this study are turret, shoulder, elbow1, elbow2 and tilt angles as shown in Figure 3.2.

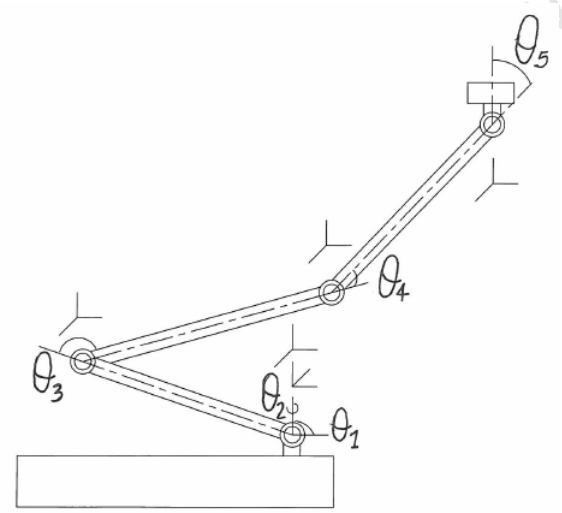


Figure 3.2: Packbot Manipulator Model, θ_1 = turret, θ_2 = shoulder, θ_3 = elbow1, θ_4 = elbow2 and θ_5 = tilt

Table 3.1 shows the measured limits of each joint. These limits are imposed by the iRobot software driver already installed on the system. The shoulder joint is restricted to a maximum of 151.97 degrees so that it does not hit objects on the base. Unfortunately, robot dimensions measured such as the length of each manipulator link and masses could not be published in this document as it could violate the agreement between the CSIR and robot manufacturer, iRobot Corporation. Each link was referred to as l_i and each mass of the link was referred to as m_i, where i is the number of the link.

Joint	Label	Min	Max
1	θ_1	0.00	360.00
2	θ_2	0.06	151.97
3	θ_3	-174.60	170.63
4	θ_4	-174.00	171.89
5	θ_5	-180.00	150.00

Table 3.1: Joints Limits. All angles measured in degrees.

It is important to note that the arm also consists of joint offsets as depicted by Figure 3.3. These offsets need to be included when computing the forward and inverse kinematics of the robot. Failing to do this would result in the position of the end-effector offset from the required end position. This will be further explained in section 3.3.

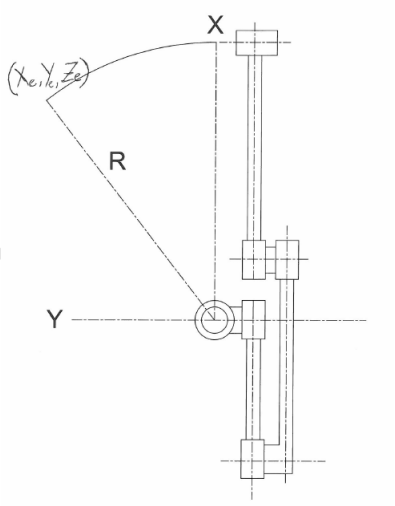


Figure 3.3: Joint Offsets. Shown from the top view.

3.2 Forward Kinematics

This section focuses on computing the forward kinematics for the Packbot manipulator. Firstly, steps followed when attaching frames to joints using the DH convention are explained. The forward kinematics solution for the Packbot manipulator is derived afterwards.

The following steps are followed when attaching frames to a joint using this convention (refer to Figure 3.4):

- Number the joints from 1 to n starting with the base and ending with the end-effector.
- Establish the base coordinate system. Establish a right-handed orthonormal coordinate system (X_0, Y_0, Z_0) at the supporting base with Z_0 axis lying along the axis of motion of joint 1.
- Establish joint axis. Align the Z_i with the axis of motion (rotary or sliding) of joint $i + 1$.
- Establish the origin of the ith coordinate system. Locate the origin of the ith coordinate at the intersection of the Z_i and Z_{i-1} or at the intersection of common normal between the Z_i and Z_{i-1} axes and the Z_i axis.
- Establish X_i axis. Establish $X_i = \pm(Z_{i-1} \times Z_i) / \parallel Z_{i-1} \times Z_i \parallel$ or along the common normal between the Z_{i-1} and Z_i axes when they are parallel.
- Establish Y_i axis. Assign $Y_i = +(Z_i \times X_i) / \parallel Z_i \times X_i \parallel$ to complete the right-handed coordinate system.
- Find the link and joint parameters.

The homogeneous transformation matrix between frame i and frame $i + 1$ is computed until the last frame and the transformation matrix that expresses the pose of the end-effector with respect to the base frame is:

$$T = A_0 A_i A_{i+1} ... A_n \tag{3.1}$$

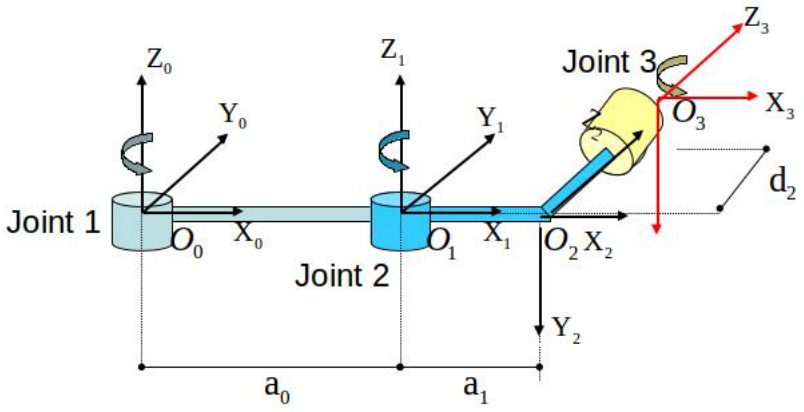


Figure 3.4: Attaching frames to joints

The *homogeneous transformation matrix* A can be computed by the following matrix multiplication:

$$A = \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & 0 \\ sin\theta_i & cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha_i & -sin\alpha_i & 0 \\ 0 & sin\alpha_i & cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$A = \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i cos\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

In Eq. (3.2) it can be seen that the first matrix on the left is a rotation about the z-axis by θ, the second one is a translation by d_i along the z-axis, the third one is the translation by a_i along the x-axis and the last one is the rotation about the x-axes by α. The parameter a is the distance between the axes z_{i-1} and z_i, and is measured along the x_i. The angle α is the angle between the axes z_{i-1} and z_i, measured in the plane normal to x_i. The positive sense for α is determined from z_{i-1} to z_i by the right-hand rule. The parameter d is the offset of the frame z_i from frame z_{i-1} along the z_i. Finally, θ is the angle between x_{i-1} and x_i measured in a plane normal z_{i-1}. The DH convention assumes that the direction of motion

of a joint is about the z-axis for a revolute and along the z-axis for a prismatic joint.

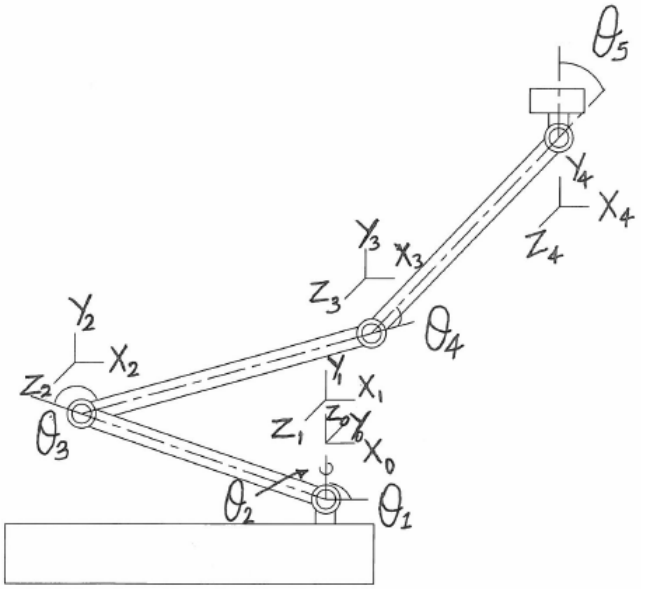


Figure 3.5: Attaching Frames To Packbot Joints. The Pack bot manipulator consists of only revolute joints therefore $q = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5)$.

Looking at Figure 3.5 the DH parameters are as follows:

Link	a_i	α_i	d_i	θ_i
1	0	90	0	θ_1
2	l_1	0	d_1	θ_2
3	l_2	0	d_2	θ_3
4	l_3	0	d_3	θ_4
5	d_6	0	0	θ_5
6	0	-90	0	0

Table 3.2: DH Parameters For Packbot Manipulator.

The corresponding transformation matrices A_i using the DH parameters from

Table 3.2 are:

$$A_1 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$A_2 = \begin{bmatrix} c_2 & -s_2 & 0 & l_1c_2 \\ s_2 & c_2 & 0 & l_1s_2 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$A_3 = \begin{bmatrix} c_3 & -s_3 & 0 & l_2c_3 \\ s_3 & c_3 & 0 & l_2s_3 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

$$A_4 = \begin{bmatrix} c_4 & -s_4 & 0 & l_3c_4 \\ s_4 & c_4 & 0 & l_3s_4 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$A_5 = \begin{bmatrix} c_5 & -s_5 & 0 & d_6c_5 \\ s_5 & c_5 & 0 & d_6s_5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

$$A_6 = \begin{bmatrix} c_6 & 0 & -s_6 & 0 \\ s_6 & 0 & c_6 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

s_i and c_i refer to $sin\theta_i$ and $cos\theta_i$ respectively. The transformation matrix from the base to the end-effector is computed as below.

$$T = A_1A_2A_3A_4A_5A_6 \tag{3.10}$$

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

where

$$r_{11} = c_1(c_2(c_3(c_4c_5 - s_4s_5) - s_3(c_4s_5 + c_5s_4)) - s_2(s_3(c_4c_5 - s_4s_5) + c_3(c_4s_5 + c_5s_4))) \tag{3.12}$$

$$r_{12} = c_1(c_2(c_3(-c_4s_5 - c_5s_4) - s_3(c_4c_5 - s_4s_5)) - s_2(c_3(c_4c_5 - s_4s_5) + s_3(-c_4s_5 - c_5s_4))) \tag{3.13}$$

$$r_{13} = s_1 \tag{3.14}$$

$$r_{14} = c_1(-s_2(s_3(-d_6s_5s_4 + l_3c_4 + c_4d_6c_5) + c3(d_6c_5s_4 + l_3s_4 + c_4d_6s_5) + l_2s_3) +$$

$$c_2(c_3(-d_6s_5s_4 + l_3c_4 + c_4d_6c_5) - s_3(d_6c_5s_4 + l_3s_4 +$$

$$c_4d_6s_5) + l_2c_3) + l_1c_2) + (d_3 + d_2 + d_1)s_1 \tag{3.15}$$

$$r_{21} = s_1(c_2(c_3(c_4c_5 - s_4s_5) - s_3(c_4s_5 + c_5s_4)) - s_2(s_3(c_4c_5 - s_4s_5) + c_3(c_4s_5 + c_5s_4))) \tag{3.16}$$

$$r_{22} = s_1(c_2(c_3(-c_4s_5 - c_5s_4) - s_3(c_4c_5 - s_4s_5)) - s_2(c_3(c_4c_5 - s_4s_5) + s_3(-c_4s_5 - c_5s_4))) \tag{3.17}$$

$$r_{23} = -c_1 \tag{3.18}$$

$$r_{24} = s_1(-s_2(s_3(-d_6 s_5 s_4 + l_3 c_4 + c_4 d_6 c_5) + c_3(d_6 c_5 s_4 + l_3 s_4 + c_4 d_6 s_5) + l_2 s_3) +$$

$$c_2(c_3(-d_6 s_5 s_4 + l_3 c_4 + c_4 d_6 c_5) - s_3(d_6 c_5 s_4 + l_3 s_4 + c_4 d_6 s_5) + l_2 c_3)$$

$$+ l_1 c_2) - c_1(d_3 + d_2 + d_1) \tag{3.19}$$

$$r_{31} = c_2(s_3(c_4 c_5 - s_4 s_5) + c_3(c_4 s_5 + c_5 s_4)) + s_2(c_3(c_4 c_5 - s_4 s_5) - s_3(c_4 s_5 + c_5 s_4)) \tag{3.20}$$

$$r_{32} = s_2(c_3(-c_4 s_5 - c_5 s_4) - s_3(c_4 c_5 - s_4 s_5)) + c_2(c_3(c_4 c_5 - s_4 s_5) + s_3(-c_4 s_5 - c_5 s_4)) \tag{3.21}$$

$$r_{33} = 0 \tag{3.22}$$

$$r_{34} = c_2(s_3(-d_6 s_5 s_4 + l_3 c_4 + c_4 d_6 c_5) + c_3(d_6 c_5 s_4 + l_3 s_4 + c_4 d_6 s_5) + l_2 s_3) +$$

$$s_2(c_3(-d_6 s_5 s_4 + l_3 c_4 + c_4 d_6 c_5) - s_3(d_6 c_5 s_4 + l_3 s_4 + c_4 d_6 s_5) +$$

$$l_2 c_3) + l_1 s_2 \tag{3.23}$$

Using the transformation matrix T, given the 5 joint angles the position of the end-effector can be obtained. Simulation results are shown in section 6.2 and experimental results in section 7.2.

3.3 Inverse Kinematics

As mentioned before, the inverse kinematics problem involves determining the joint angles of the manipulator necessary to bring the end-effector to a specific

position and orientation. In the case of this manipulator type the joint angles that need to be calculated are θ_1, θ_2, θ_3, θ_4 and θ_5 as shown in Figure 3.2 and their limits shown in table 3.1. The first thing that needs to be done is to work out the manipulator workspace. The manipulator's total length when fully stretched is defined by $l = l_1 + l_2 + l_3$ where l_1, l_2 and l_3 are the lengths of the three links and l is the total manipulator length. See Figure 3.6 for the manipulator workspace.

The arm is restricted to spaces above the base as it would hit the ground if allowed to go below the base. One might argue that the arm should be allowed to reach below the base since the Packbot has flippers to lift it up in the air. This might be useful for picking up objects or self inspection, however, the aim of this study is to position the end-effector close to the roof. Therefore, self inspection and picking up objects are not considered although this method can be easily extended to work in those cases. Any point outside the semi circle above the base with radius l centered at the origin is also considered unreachable as the arm cannot reach beyond the semi-circle. Areas inside the circle with radius l_1 centered at the origin have also been restricted because it is very difficult for the end-effector to reach there. Lastly, the off-sets on the joints pose a problem when the arm tries to reach directly above the base frame. As a result of that, a cylinder of radius d centered at the base frame as shown in Figure 3.6 is also unreachable. d is the total offset of the arm as shown in Figure 3.7. The reachable space is represented by the shaded area in Figure 3.6. If $l_1 < R_e < l_1 + l_2 + l_3$ is satisfied then we have a solution to the inverse kinematics problem. R_e is the euclidean distance from the origin to the goal position.

The first step is to calculate θ_1. Given the end-effector goal position as (x_e, y_e, z_e), the distance from the origin to the position of the end-effector is given by:

$$R_e = \sqrt{{x_e}^2 + {y_e}^2 + {z_e}^2} \tag{3.24}$$

The angle θ_1 can be calculated using the arctan of the X and Y coordinates of the end-effector goal position. However, different quadrants yield different results. To make sure that the correct results are found in the correct quadrant the following modifications are made.

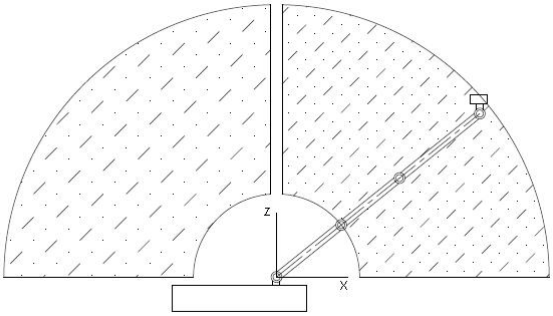


Figure 3.6: Robot Workspace in XZ plane

First quadrant (Both x_e and y_e are positive).

$$\theta_1 = \arctan \frac{y_e}{x_e} \tag{3.25}$$

Looking at the structure of the manipulator, if you were to apply θ_1 calculated above, the end-effector would end up slightly offset from the goal position as shown on the figure below.

The solution to this is to calculate the sum of the offsets d and use it to calculate the offset angle then add it to or subtract it from θ_1 depending on the given quadrant. In this quadrant the actual θ_1 would be calculated in this manner.

$$\rho = \arctan \frac{d}{R_e} \tag{3.26}$$

$$\theta_1 = \arctan \frac{y_e}{x_e} + \rho \tag{3.27}$$

Another solution for θ_1 would be

$$\theta_1 = 180 + \arctan \frac{y_e}{x_e} + \rho \tag{3.28}$$

Second quadrant (x_e is negative and y_e is positive)

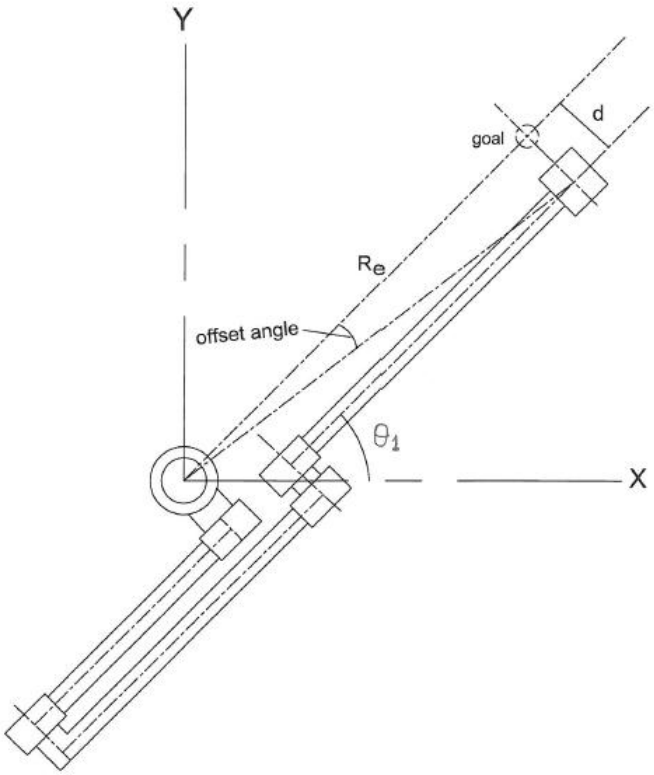


Figure 3.7: θ_1 offset

$$\theta_1 = \arctan \frac{y_e}{x_e} - \rho \tag{3.29}$$

or

$$\theta_1 = 180 - \arctan \frac{y_e}{x_e} + \rho \tag{3.30}$$

Third quadrant (Both x_e and y_e are negative).

$$\theta_1 = \arctan \frac{y_e}{x_e} + \rho \tag{3.31}$$

or

$$\theta_1 = 180 + \arctan \frac{y_e}{x_e} + \rho \tag{3.32}$$

Fourth quadrant (x_e is positive and y_e is negative)

$$\theta_1 = \arctan\frac{y_e}{x_e} - \rho \tag{3.33}$$

or

$$\theta_1 = 180 + \arctan\frac{y_e}{x_e} - \rho \tag{3.34}$$

If $x_e = y_e = 0$ and $z_e > l_1$ the θ_1 angle can be calculated as shown below.

$$\theta_1 = 0 + \rho \tag{3.35}$$

or

$$\theta_1 = 180 + \rho \tag{3.36}$$

If $x_e = 0$ and $y \neq 0$

$$\theta_1 = -90 + \rho \tag{3.37}$$

or

$$\theta_1 = 90 + \rho \tag{3.38}$$

If $y_e = 0$ and $x_e \neq 0$

$$\theta_1 = 0 + \rho \tag{3.39}$$

or

$$\theta_1 = 180 + \rho \tag{3.40}$$

The idea behind calculating θ_2 is to project the goal position coordinates to the global XZ plane, calculate θ_2 in the 2D plane then project it back (See Figure 3.8). There are multiple solutions for θ_2 but not infinite. The idea is to calculate the minimum and maximum values that will allow the end-effector to reach the desired goal. Any angle between the two extremes is a possible solution.

After projecting the goal position to the XZ plane using the following calculations:

$$R = \sqrt{{x_e}^2 + {y_e}^2} \tag{3.41}$$

$$X_n = \sqrt{(R)^2 - (d)^2} \tag{3.42}$$

X_n is the new x axis in the XZ plane.

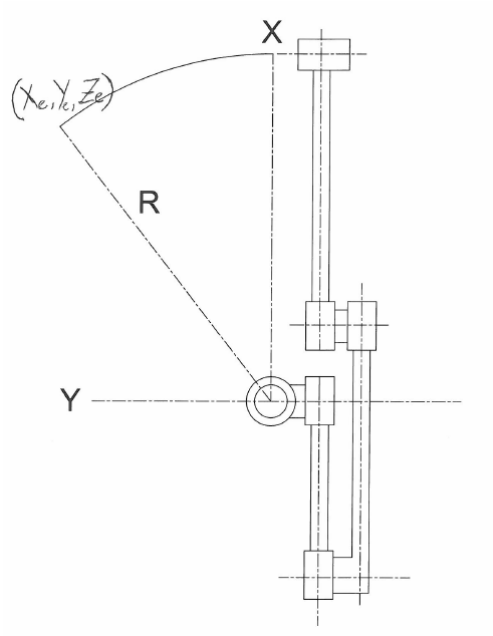


Figure 3.8: Goal projected to XZ plane

The θ_2 range is calculated by finding the two intersections between a circle centered at the goal position with radius $l_2 + l_3$ and the circle centered at the origin with radius l_1. See Figure 3.9. Using the intersection (x_{int}, z_{int}) found, the min θ_{2min} and max θ_{2max} can be computed using the following formula:

If x_{int} and z_{int} are on the first quadrant

$$\theta_2 = \arctan \frac{z_{int}}{x_{int}} \tag{3.43}$$

or if they are on the second quadrant

$$\theta_2 = 180 - \arctan \frac{z_{int}}{-x_{int}} \tag{3.44}$$

θ_2 is the minimum of the two and θ_{2max} is the maximum. Every angle between

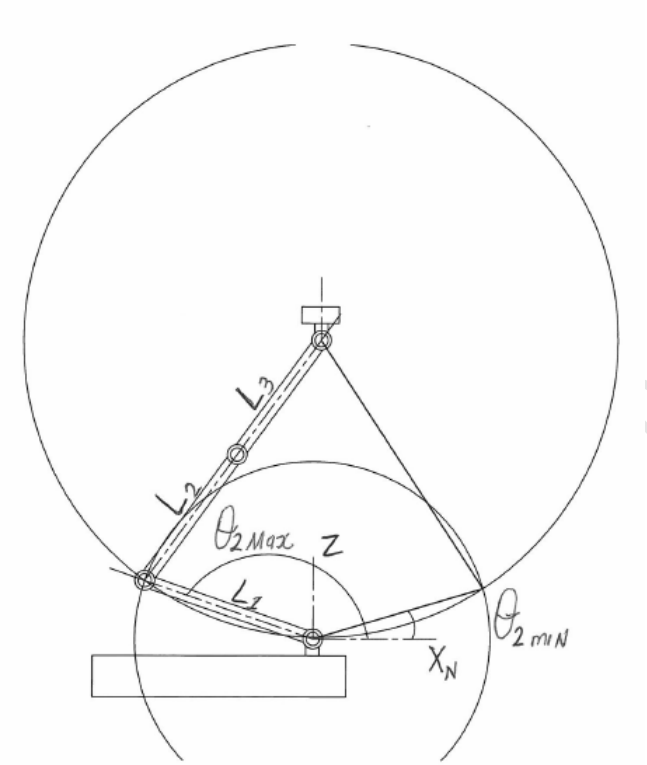


Figure 3.9: θ_2 angle

θ_{2min} and θ_{2max} can be used as θ_2. Therefore,

$$\theta_{2min} =< \theta_2 <= \theta_{2max} \tag{3.45}$$

Given θ_2 angle calculated above, θ_3 angle can be calculated. Using the forward kinematics of the robot and putting θ_1 and θ_2 angles in their corresponding positions in the transformation matrix will yield the x, y, z coordinates of the end of link 1 (x_2, y_2, z_2 of T_3).

Given those points θ_3 joint angle can be calculated by finding the intersection between the circle centered at the goal position with radius l_3 and the circle centered at x_2, y_2, z_2 with radius l_2 as shown in Figure 3.10 .

Using the intersection found above (x_{int2}, z_{int2}) the distance S can be computed by the distance formula.

$$S = \sqrt{(x_{int2} - x_b)^2 + (z_{int2} - z_b)^2} \tag{3.46}$$

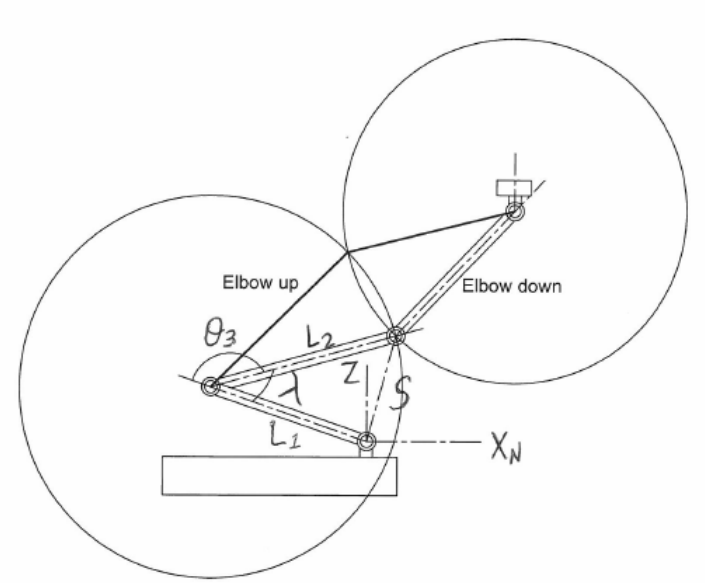


Figure 3.10: θ_3 angle

With x_b and z_b representing base frame or origin coordinates.

The angle λ can be calculated using the law of cosines as follows:

$$\lambda = \arccos \frac{({l_1}^2 + {l_2}^2 - s^2)}{(2l_1 l_2)} \tag{3.47}$$

Using λ calculated above, θ_3 angle can be calculated.

$$\theta_3 = \pm(180 - \lambda) \tag{3.48}$$

The sign of θ_3 is measured about the z_2 axis as shown in Figure 3.1. There are always two solutions, one for elbow up and one for elbow down configurations.

The angle θ_4 is computed using Figure 3.11 and the following calculations.

Distance t can be found using the distance formula.

$$t = \sqrt{(x_n - x_3)^2 + (z_e - z_3)^2} \tag{3.49}$$

Using the law of cosines, ϕ can be calculated.

$$\phi = \arccos \frac{({l_2}^2 + {l_3}^2 - t^2)}{(2l_2 l_3)} \tag{3.50}$$

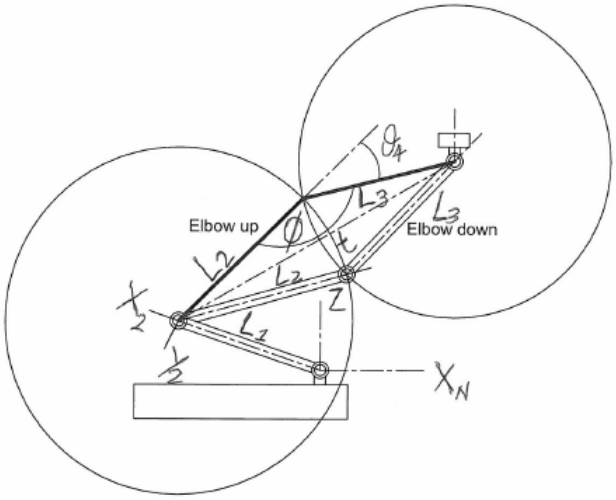


Figure 3.11: θ_4 angle

Finally, θ_4 can be calculated as follows,

For elbow up configuration

$$\theta_4 = -(180 - \phi) \tag{3.51}$$

For elbow down configuration

$$\theta_4 = 180 - \phi \tag{3.52}$$

In order to have the sensor represented by the rectangular box at the end-effector at the correct position, z_e used to calculate the previous joints should be the sensor's Z component minus d_6 ($z_e = Z - d_6$). d_6 is the link connecting link 3 to the end-effector. θ_5 can be found by first calculating σ as shown in figure 9. σ can be found using the law of cosines.

$$\sigma = \arccos \frac{({l_3}^2 + {d_6}^2 - q^2)}{(2l_3 d_6)} \tag{3.53}$$

The distance q is found using the distance rule.

$$q = \sqrt{(Z - z_3)^2 + (x_e - x_3)^2} \tag{3.54}$$

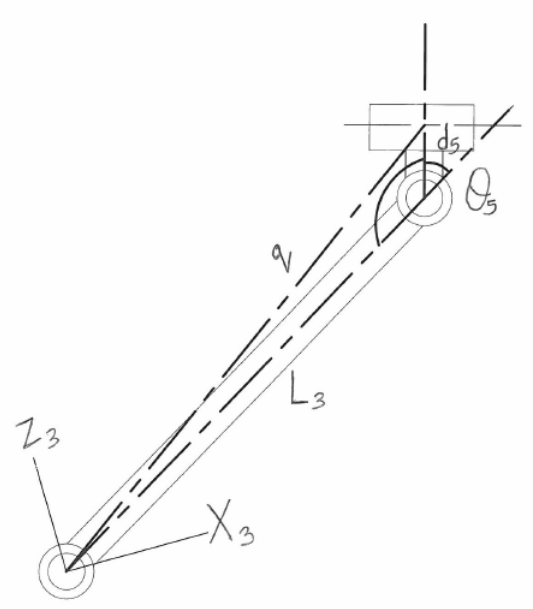


Figure 3.12: θ_5 angle

$$\theta_5 = \pm(180 - \sigma) \tag{3.55}$$

Algorithm 1 shows how this inverse kinematics method was implemented. The work done in this section was published in [72] and simulation results showcasing the performance of this algorithm are shown in Chapter 6 and experimental results to follow in Chapter 7.

Algorithm 1 IK Algorithm

1: Get the desired end-effector position
2: Check if it is reachable
3: **if** Goal reachable **then**
4: continue below
5: **else**
6: go to line 1
7: **end if**
8: Calculate θ_1
9: Calculate the range of θ_2
10: **for** $\theta_{2min} \leftarrow$ to θ_{2max} **do**
11: Calculate θ_3
12: Calculate θ_4
13: Calculate θ_5
14: **end for**

Chapter 4

Planning for the Packbot510i Manipulator

This chapter deals with the Rapidly exploring Random Trees algorithms used in this study without tip-over avoidance. It is divided into three sections, the first section explains how the basic RRT algorithm is constructed, an extension to the basic RRT called RRT ball is explained in section 4.2 and finally, the RRT* is dealt with in section 4.3. The reader is referred to [54] for a deeper understanding of RRT and [52] for RRT Ball and RRT*. On each section, a 2D arm example of each method is provided in order to easily visualize the trees and understand how they can be implemented and they will be extended to work with a 5 DOF Packbot510i manipulator.

The following 2D arm shown in Figure 4.1 will be used as an example. Given the initial end-effector position (x_i, y_i) and the goal position (x_g, y_g) in a Cartesian plane, the goal is to take the end-effector from (x_i, y_i) to (x_g, y_g). Planning is done in joint space in this case, i.e, instead of looking for the x and y values of the end-effector that will drive the arm to the goal position, we look for θ_1 and θ_2 values. That means we have to transform the initial and goal end-effector position to their corresponding (θ_1, θ_2) using inverse kinematics. That leaves us with the initial configuration $q_{init} = (\theta_{1i}, \theta_{2i})$ and the goal configuration $q_{goal} = (\theta_{1g}, \theta_{2g})$ and our configuration space consists of all the (θ_1, θ_2) pairs in the 2D space. The function of the RRT is to find a set of these configurations, which are also called

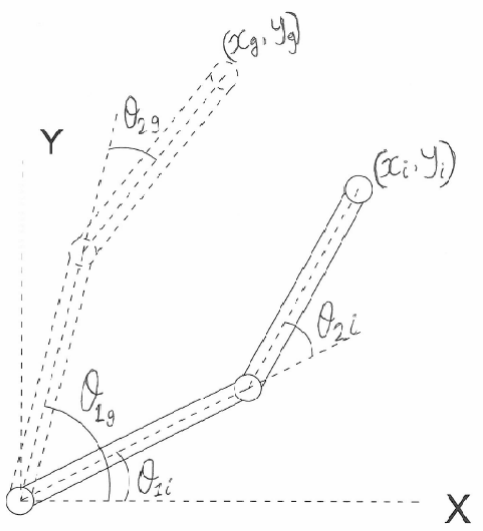


Figure 4.1: 2D Manipulator

nodes, from the initial to the goal configuration.

4.1 Rapidly exploring Random Trees (RRT)

The basic RRT algorithm is summarized in Algorithm 2 and explained below. After initializing the tree T with q_{init}, a simple iteration is performed in which each step attempts to extend the RRT by adding a new vertex that is biased by a randomly-selected node q_{rand} using RANDOM_CONFIG. If q_{rand} is in collision then it is ignored and a new q_{rand} is generated. The algorithm then looks for the nearest neighbor q_{near} of q_{rand} already in the tree using NEAREST_NEIGHBOR. The q_{near} is expected to be collision-free since it has already been stored in the tree. The *EXTEND* function, illustrated in Figure 4.2, makes a motion toward q_{rand} with some fixed incremental distance ϵ, tests for collision and makes q_{near} the parent of q_{new}. This process is performed successively until the sampled node finally reaches the goal, where the route connecting the points is called a "tree".

Once the goal node has been reached, a path needs to be traced back to the start node. This is done by first finding the node closest to the goal or at the goal and look for its parent in the tree and trace it back until we reach the start node. This process is illustrated in Figure 4.3.

Algorithm 2 Basic RRT construction Algorithm

BUILD_RRT(q_{init})

```
1: T.init(q_init);
2: for k = 1 to K do
3:    q_rand ← RANDOM_CONFIG();
4:    if q_rand ← COLLISION then
5:       CONTINUE;
6:    end if
7:    q_near ← NEAREST_NEIGHBOR();
8:    EXTEND(T,q_rand, q_near);
9: end for
10: Return T
```

EXTEND(T,q,q_{near})

```
1: T.add_vertex(q_new);
2: T.add_edge(q_near,q_new);
3: if q_new ← COLLISION then
4:    CONTINUE;
5: end if
6: Return T;
```

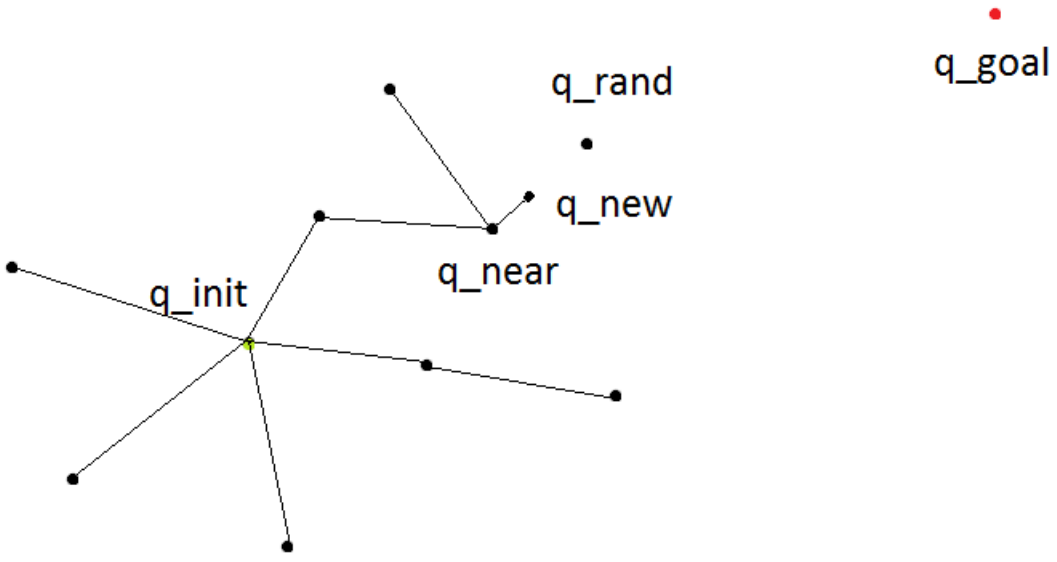



Figure 4.2: The Extend function

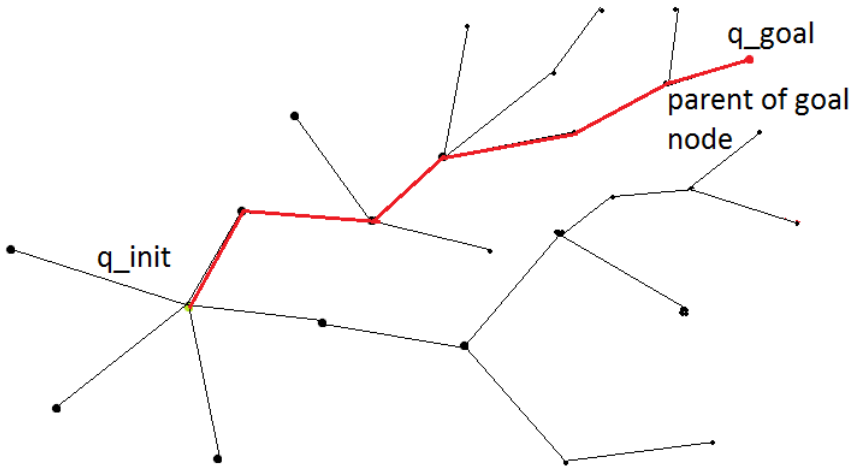


Figure 4.3: Tracing back the path

Figure 4.4 shows a tree after 2303 iterations for $q_{init} = (0, 0)$ and $q_{goal} = (0.9, 0.9)$. The tree was immediately terminated after finding the goal. Figure 4.5, 4.6, 4.7 and 4.8 show the trees for N = 5 000, N = 10 000, N = 50 000 and N = 100 000 iterations respectively.

In Figure 4.4, the goal was found after 2303 iterations and the RRT did not fill the entire space as it was terminated immediately after finding the goal. The RRT took CPU time of < 0 ms, i.e., it could not be measured by the timer and the cost of the path was 1.5794 radians. The unit is in radian since this RRT planning for a 2DOF manipulator is in joint space. CPU time in this case is the amount of time in seconds taken for the central processing unit to process the algorithm and when the time taken is too small and cannot be measured by the timer it reports 0 ms so the notation "$<$ 0 ms" is used to represent that throughout the document. In Figure 4.5, the tree was allowed to expand until $N = 5000$ iterations have been reached. In this case the tree filled more space compared to 4.4 and it took CPU time of < 0 ms as well with a path cost of 1.7922 radians. For $N = 10000$, shown in Figure 4.6, the tree covered more space compared to the previous two cases. It took CPU time of < 0 ms with a path cost of 1.6535 radians. For N = 50 000 in Figure 4.7 it took CPU time of 3000 ms (3 seconds) with a path cost of 1.6219 radians and for N = 100 000 in Figure 4.8 it took CPU time of 14 000 ms (14 seconds) with a path cost of 1.5837 radians.

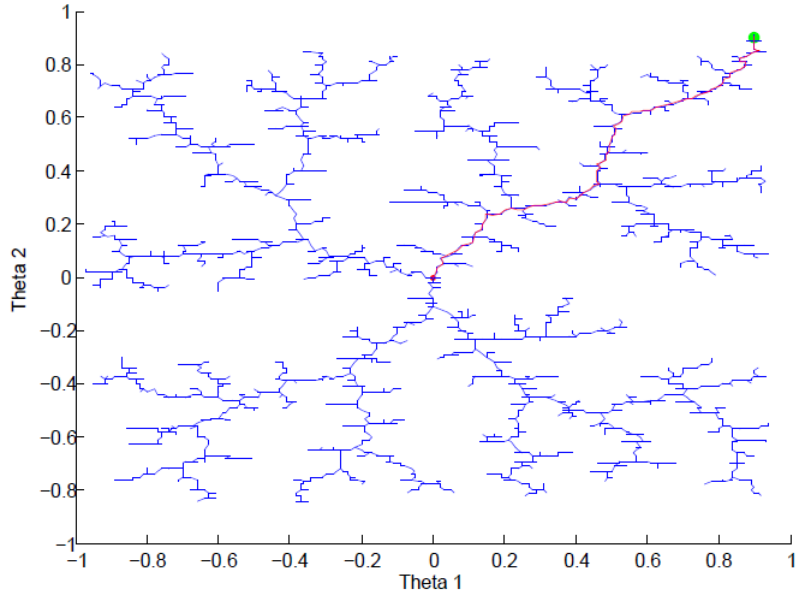


Figure 4.4: Basic RRT tree after 2303 iterations. Cost of the path is 1.5794 radians. CPU time < 0 ms.

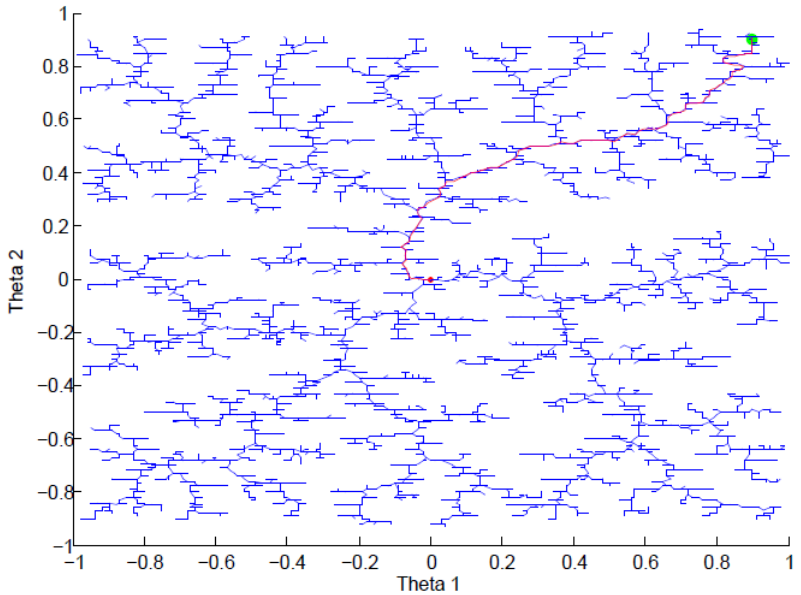


Figure 4.5: Basic RRT tree after 5000 iterations. Cost of the path is 1.7922 radians. CPU time < 0 ms.

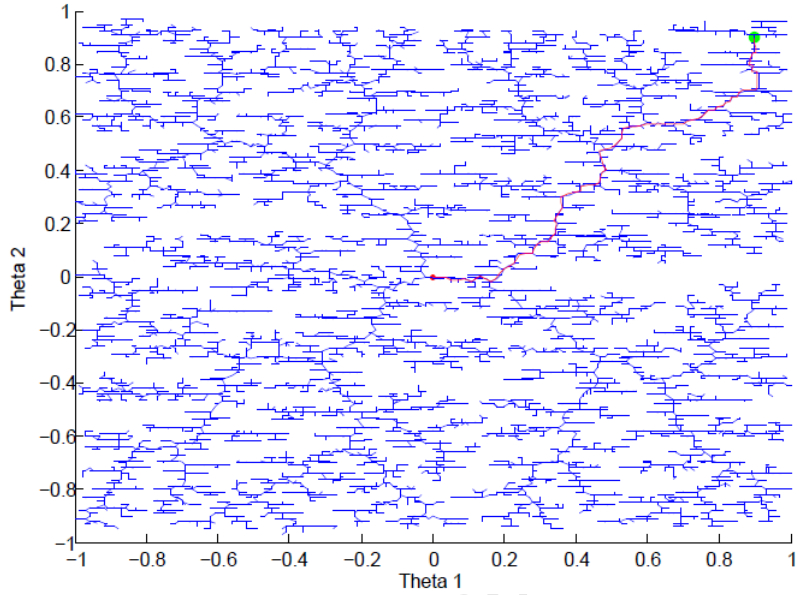


Figure 4.6: Basic RRT tree after 10000 iterations. Cost of the path is 1.6535 radians. CPU time < 0 ms.

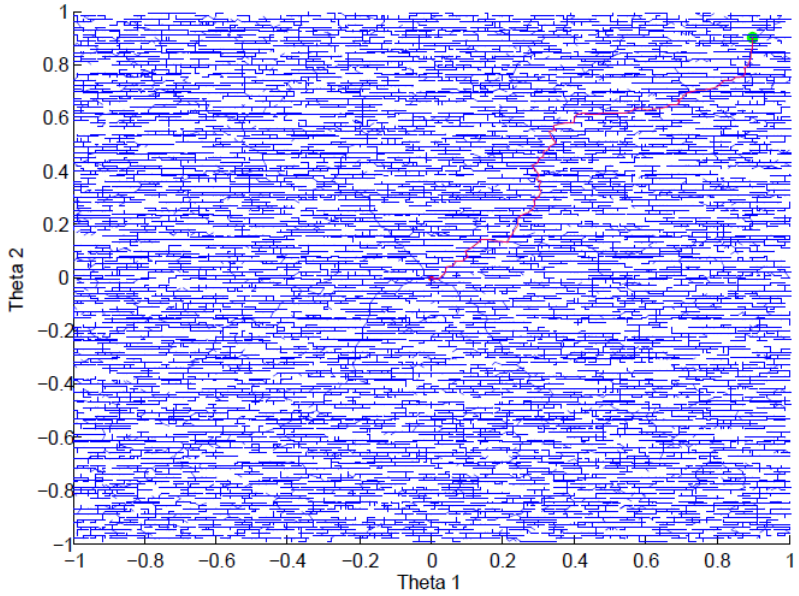


Figure 4.7: Basic RRT tree after 50000 iterations. Cost of the path is 1.6219 radians. CPU time is 3000 ms (3 seconds).

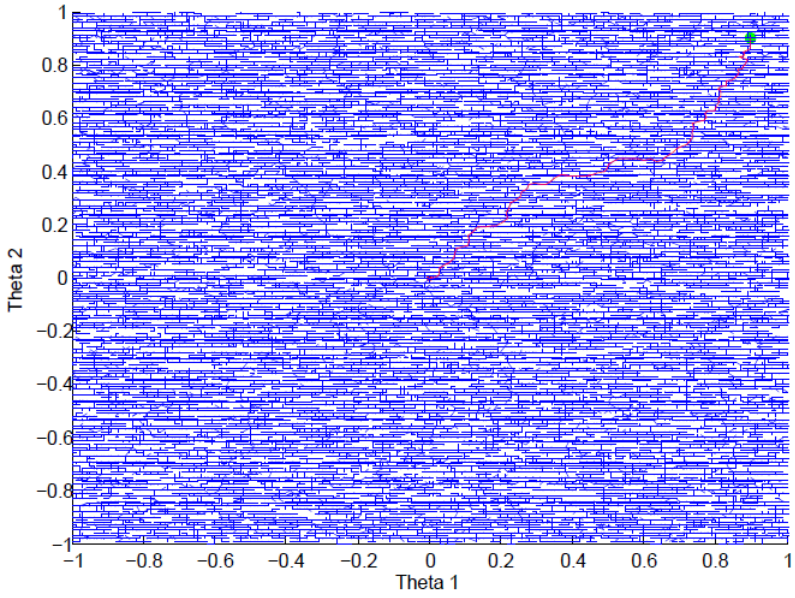


Figure 4.8: Basic RRT tree after 100000 iterations. Cost of the path is 1.5837 radians. CPU time is 140000 ms (14 seconds).

Looking at Figure 4.4 to 4.8, the tree seems to cover more space as the number of iterations increases as expected. The number of nodes populated in the space increases as more time is available to grow the tree. Now, since the basic RRT algorithm does not consider the notion of cost of the path, the cost of the path produces does not seem to decrease as more time is given to the planner.

In fact, increasing the number of iterations increases the computational time, for example, it takes 14 seconds CPU time for N = 100 000 with a path cost of 1.5837 radians and takes < 0 ms for N = 2303 with a path cost of 1.5794 radians. This shows that running this algorithm for a longer period of time gives you more coverage of the environment but does not guarantee an optimal path in terms of distance from the start to the end. This is also due to the fact that the RRT algorithm is biased towards unexplored regions and not the goal. Generally you would expect that an algorithm that is given more time to explore the environment would find a better path because it has more time to explore all possible ways to get to the goal. This means that a more optimal algorithm that takes the path cost into consideration is needed and this is where the RRT Ball and RRT* come in. The summary of the results of the RRT algorithm in Figure

4.4 to 4.8 is given in Table 4.1.

Iterations	CPU time	Cost
2303	< 0.00	1.58
5000	< 0.00	1.79
10000	< 0.00	1.65
50000	3.00	1.62
100000	14.00	1.58

Table 4.1: Summary of the RRT Algorithm. Time given in seconds and cost in radians.

4.2 Rapidly exploring Random Trees Ball (RRT ball)

The RRT ball works similar to the basic RRT with few additional features. After a new node has been added to the tree, its euclidean distance back to the start node via its parent is stored. A ball is created, with radius of $g * log(i)/i$ where i is the number of the current iteration, around the new node and all nodes already in the tree within that ball are stored. The closest node to the new node in the ball is found and removed from the tree. All other nodes in the ball are reconnected if faster. This process is called rewiring the tree. The nodes in the ball are reconnected if their distances back to start via the new node (DISTANCE(q_{new},q_i) see Algorithm 3) are less than their distances via their current parents (DISTANCE(q_{old},q_i)). If that is the case, the new node is assigned as their new parent and their distances are updated. On the second iteration, the closest node to the new node in the ball is found and removed again. This rewiring process iterates until there are no nodes left in the ball. The entire process repeats until a goal is found or N iterations have been reached. Letting the RRT ball run after an initial goal has been found to result in a more optimal path in terms of distance to the goal but increases the computational burden.

Algorithm 3 RRT Ball construction Algorithm

BUILD_RRT(q_{init})

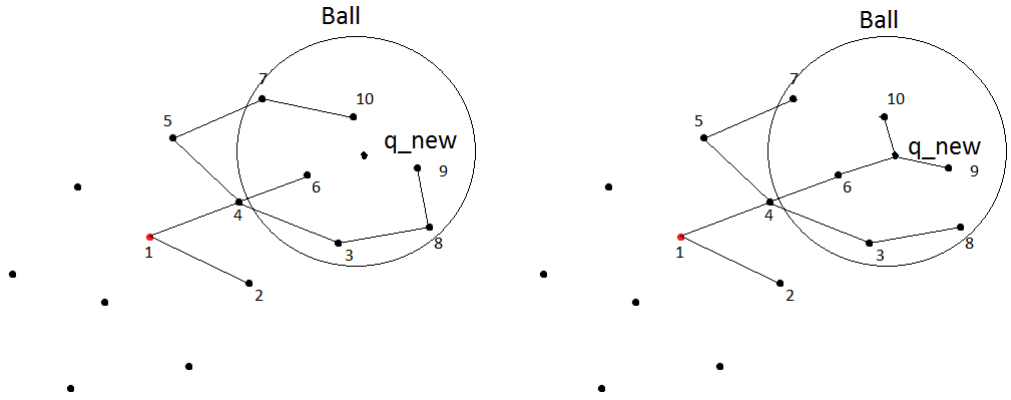
1: T.init(q_{init});
2: **for** $k = 1$ to K **do**
3: $q_{rand} \leftarrow$ RANDOM_CONFIG();
4: **if** $q_{rand} \leftarrow$ COLLISION **then**
5: CONTINUE;
6: **end if**
7: $q_{near} \leftarrow$ NEAREST_NEIGHBOR();
8: EXTEND(T,q_{rand}, q_{near});
9: REWIRE(T,q_{new});
10: **end for**
11: Return T

EXTEND(T,q,q_{near})

1: T.add_vertex(q_{new});
2: T.add_edge(q_{near},q_{new});
3: **if** $q_{new} \leftarrow$ COLLISION **then**
4: CONTINUE;
5: **end if**
6: Return T;

REWIRE(T,q)

1: CreateBall(q_{new});
2: **while** Ball.Size ¿ 0 **do**
3: $q_{near} \leftarrow$ NEAREST_NEIGHBOR();
4: REMOVE(q_{near});
5: **if** DISTANCE(q_{old},q_i) > DISTANCE(q_{new},q_i) **then**
6: PARENT(q_i) = q_{new}
7: **end if**
8: **end while**
9: Return T;



(a) Stage 1 (b) Stage 2

Figure 4.9: Rewiring Phase

Nodes 3, 6, 7, 8, 9 and 10 in Figure 4.9(a) are inside the ball therefore are chosen for reconnection. Node 6 is the parent of the new node. Now the distances of node 9 and 10 via the new node are less than their distances via their current parents (node 8 and 7 respectively). Their new parent becomes the new node and the rest of the nodes are left untouched since their distances are already the shortest. This is illustrated in Figure 4.9(a) and 4.9(b).

Figure 4.10 shows an RRT Ball tree for $q_{int} = (0, 0)$ and $q_{goal} = (0.9, 0.9)$ after 2255 iterations and the tree was terminated after the goal was found. Figure 4.11, 4.12, 4.13 and 4.14 show Ball trees for N = 5000, N = 10 000, N = 50 000 and N = 100 000 iterations respectively.

In Figure 4.10, the goal was found after 2255 iterations and the tree did not cover the entire space just like in Figure 4.4. It took RRT Ball the same CPU time of < 0 ms to find a goal as RRT did in Figure 4.4 with a lower path cost of 1.3261 radians. For N = 5000 iterations, in Figure 4.11, it took < 0 ms with a path cost of 1.3734 radians which is lower than that of RRT for the same number of iterations in Figure 4.5. The path cost in this case is not shorter than for N = 2255 for RRT Ball as would be expected and this will be analyzed after all the tests for RRT Ball have been completed. For N = 10 000 in Figure 4.12, it took CPU time of < 0 ms with a higher path cost of 1.4138 radians.

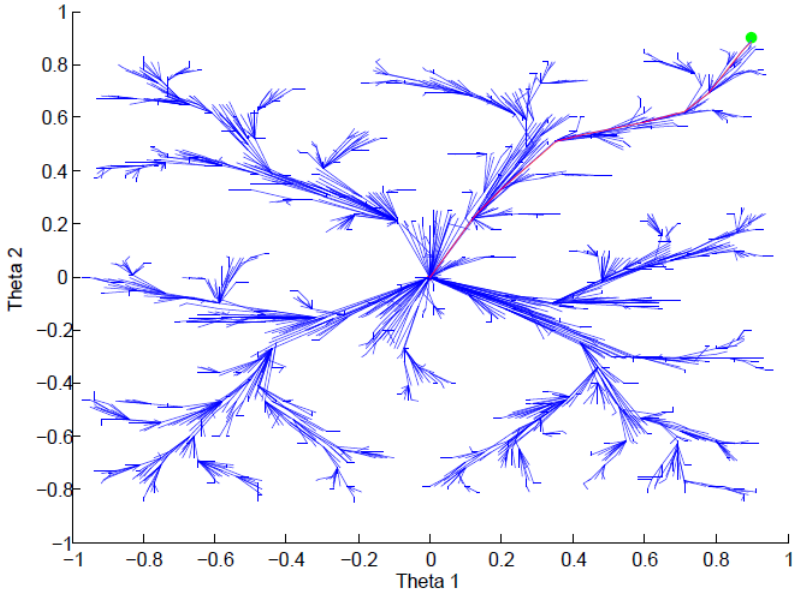


Figure 4.10: RRT Ball tree after 2255 iterations. Cost of the path is 1.3261 radians. CPU time < 0 ms.

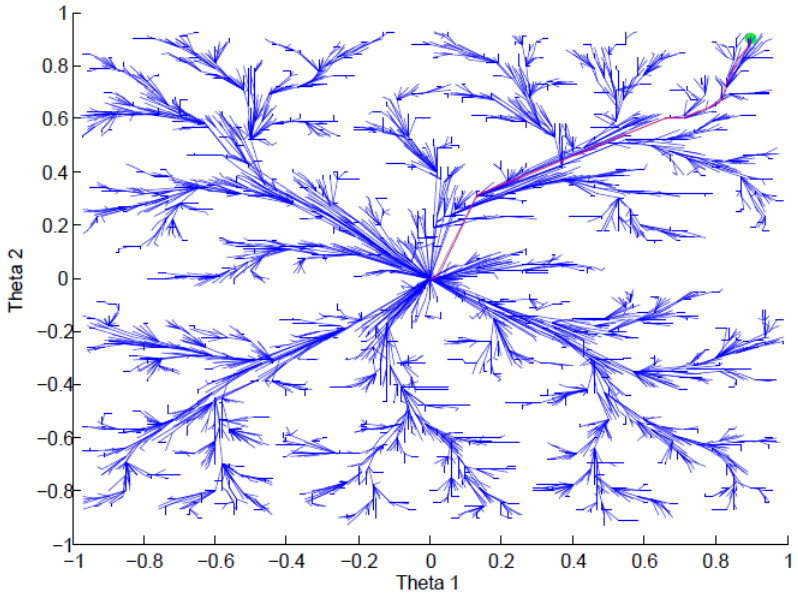


Figure 4.11: RRT Ball tree after 5000 iterations. Cost of the path is 1.3734 radians. CPU time < 0 ms.

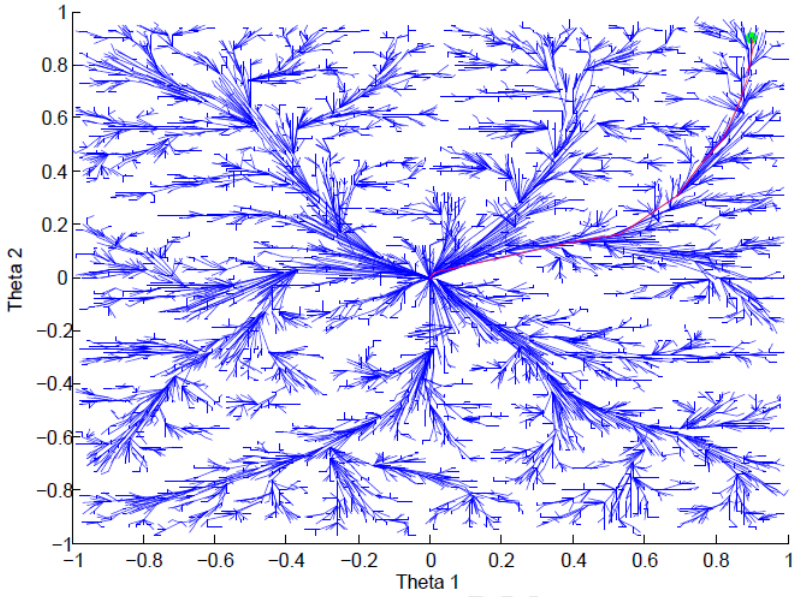


Figure 4.12: RRT Ball tree after 10000 iterations. Cost of the path is 1.4138 radians. CPU time < 0 ms.

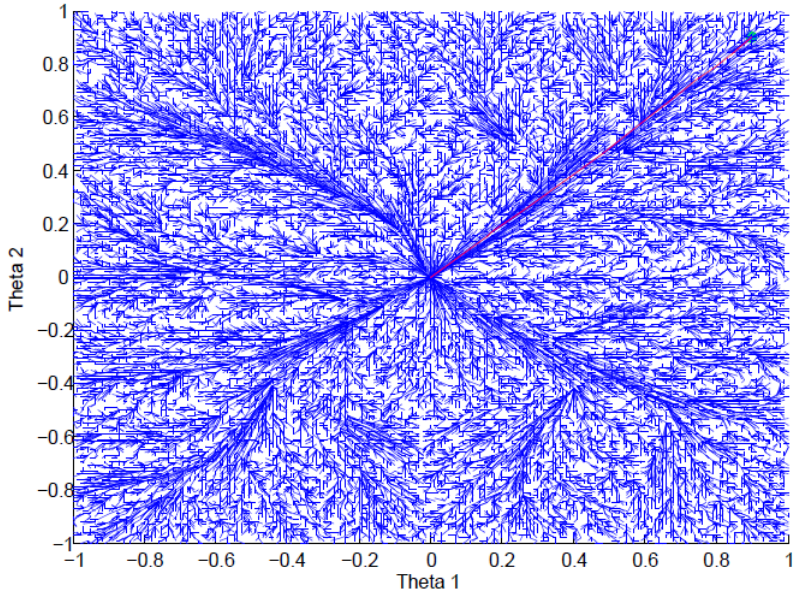


Figure 4.13: RRT Ball tree after 50000 iterations. Cost of the path is 1.2735 radians. CPU time is 7000 ms (7 seconds).

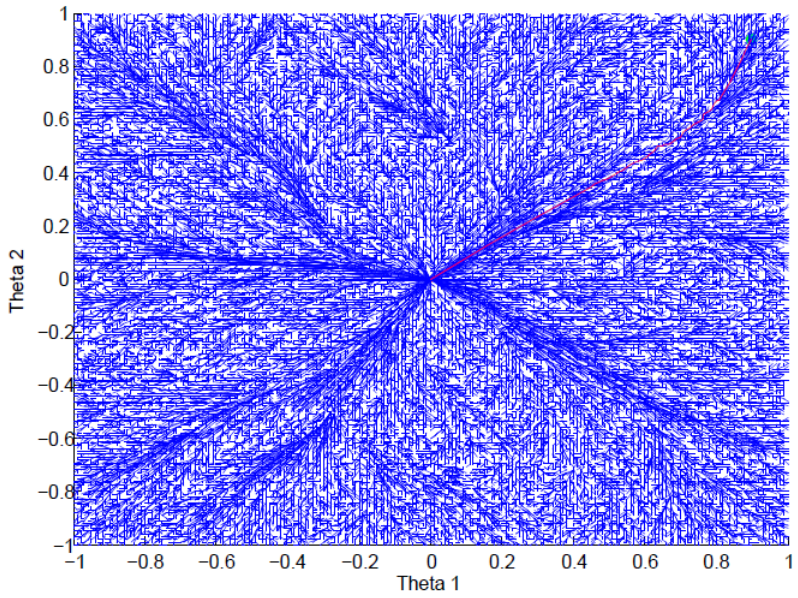


Figure 4.14: RRT Ball tree after 100000 iterations. Cost of the path is 1.3136 radians. CPU time is 40 000 ms (40 seconds).

For N = 50 000 in Figure 4.13, the CPU time increases to 7 seconds and the path cost decreases to 1.2735 radians. In this case the path seems to be an optimal straight line joining the start and the finish but it took longer to produce that path. The RRT Ball with N = 100 000 in Figure 4.14 took 40 000 ms (40 seconds) with a path cost of 1.3136 radians. RRT Ball seems to be more optimal in terms of path cost but as the number of iterations increases it becomes computationally expensive compared to RRT. The RRT Ball performs better than the RRT in terms of path cost when the number of iterations is not big, i.e, less than 10 000 but it is not consistent in terms of the cost of the path, i.e, the path cost does not always decrease as the number of iterations increase. In the section RRT Star will be explained and compared to RRT and RRT Ball. The summary of the results of the RRT Ball algorithm in Figure 4.10 to 4.14 is given in Table 4.2.

Iterations	CPU time	Cost
2255	< 0.00	1.33
5000	< 0.00	1.37
10000	< 0.00	1.41
50000	7.00	1.27
100000	40.00	1.31

Table 4.2: Summary of the RRT Ball Algorithm. Time given in seconds and cost in radians.

4.3 Rapidly exploring Random Trees Star (RRT*)

In the RRT*, before a new node can be added to the tree, the best parent to this new node is found and assigned to it by BEST_PARENT(T,q_{new}) as illustrated in Figure 4.15.

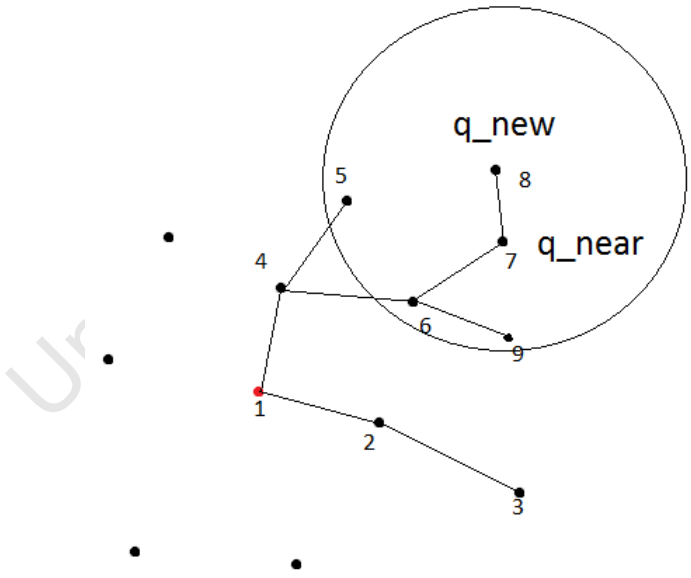
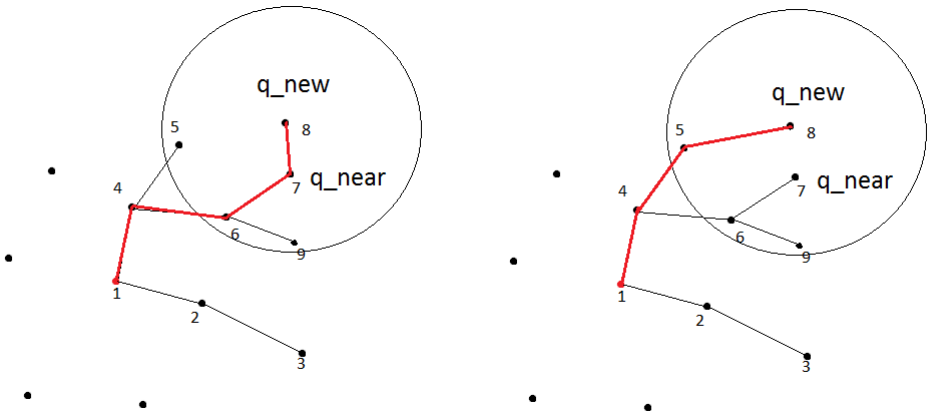


Figure 4.15: Finding the best parent

The best parent is the node whose distance from the start node and to the new node is the shortest. A ball is constructed around the new node as in RRT ball. The current best distance is the distance from the start to the node (q_{near} see Figure 4.15) which new node was extended from plus the distance from this

node to the new node. If the distance from the start to any node in the ball plus the distance from this node to the new node is less than the current best distance, then this node is the best parent to new node and is assigned as the parent of new node as shown in Figure 7.11(a) and 7.11(b).



(a) Stage 1 (b) Stage 2

Figure 4.16: Finding the best distance

New node is then added to the tree via this best parent and its distance is via the best parent. In the rewiring phase, the closest node to the new node in the ball is removed as in RRT ball. For the remaining nodes, if their distances via new node (DISTANCE(q_{new},q_i)) are less than their distances via their current parents (DISTANCE(q_{old},q_i)) then new node is assigned as their new parent as shown in Figure 4.9(a) and 4.9(b). The entire process is repeated until the goal has been found or N iterations have been exhausted.

Figure 4.17 shows an RRT* tree for $q_{int} = (0, 0)$ and $q_{goal} = (0.9, 0.9)$ after 2164 iterations and the tree was terminated after the goal was found. Figure 4.18, 4.19, 4.20 and 4.21 show RRT* trees for N = 5000, N = 10 000, N = 50 000 and N = 100 000 iterations respectively.

Algorithm 4 RRT* Construction Algorithm

BUILD_RRT(q_{init})

1: T.init(q_{init});
2: **for** $k = 1$ to K **do**
3: $q_{rand} \leftarrow$ RANDOM_CONFIG();
4: **if** $q_{rand} \leftarrow$ COLLISION **then**
5: CONTINUE;
6: **end if**
7: $q_{near} \leftarrow$ NEAREST_NEIGHBOR();
8: EXTEND(T,q_{rand}, q_{near});
9: $q_{best} \leftarrow$ BEST_PARENT(T,q_{new});
10: REWIRE(T,q_{new});
11: **end for**
12: Return T

EXTEND(T,q,q_{near})

1: T.add_vertex(q_{new});
2: T.add_edge(q_{near},q_{new});
3: **if** $q_{new} \leftarrow$ COLLISION **then**
4: CONTINUE;
5: **end if**
6: Return T;

REWIRE(T,q)

1: CreateBall(q_{new});
2: $q_{near} \leftarrow$ NEAREST_NEIGHBOR();
3: REMOVE(q_{near});
4: **for** $u = 1$ to Ball.Size **do**
5: **if** DISTANCE(q_{old},q_i) > DISTANCE(q_{new},q_i) **then**
6: PARENT(q_i) = q_{new}
7: **end if**
8: **end for**

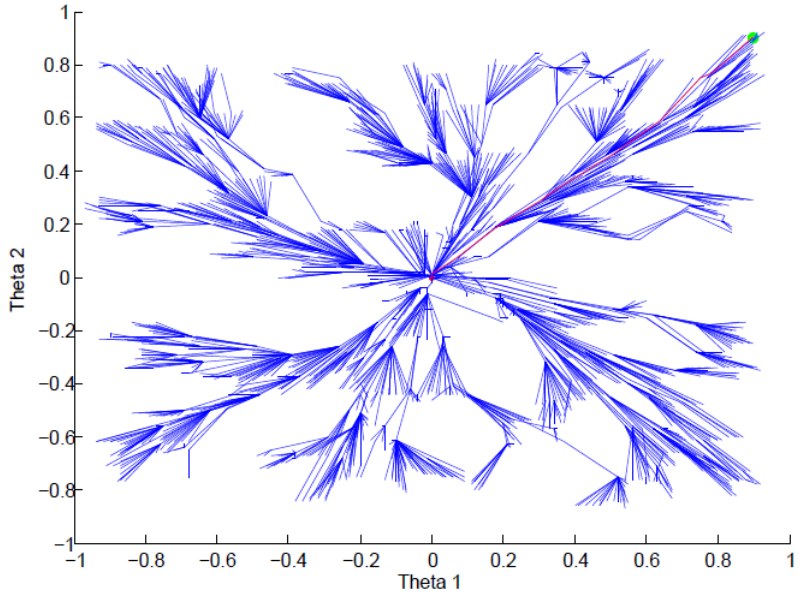


Figure 4.17: RRT* tree after 2164 iterations. Cost of the path is 1.2855 radians. CPU time < 0 ms.

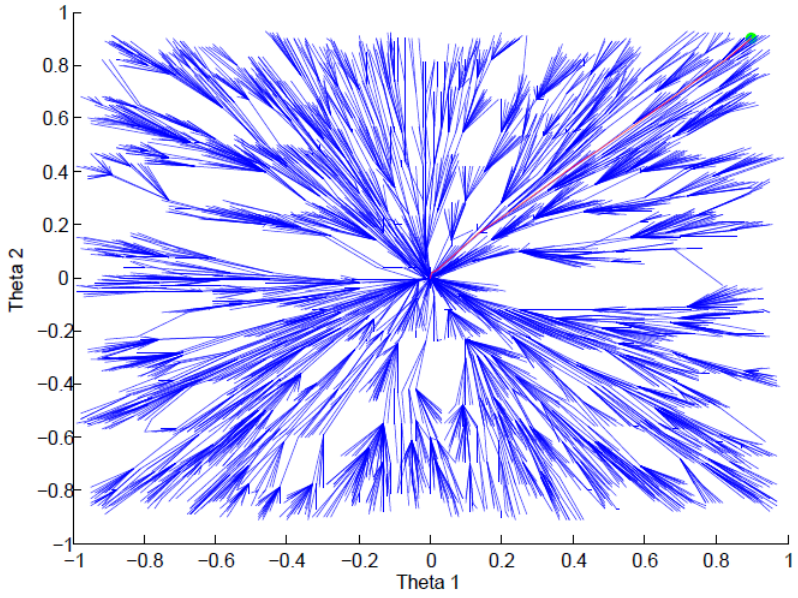


Figure 4.18: RRT* tree after 5000 iterations. Cost of the path is 1.2788 radians. CPU time < 0 ms.

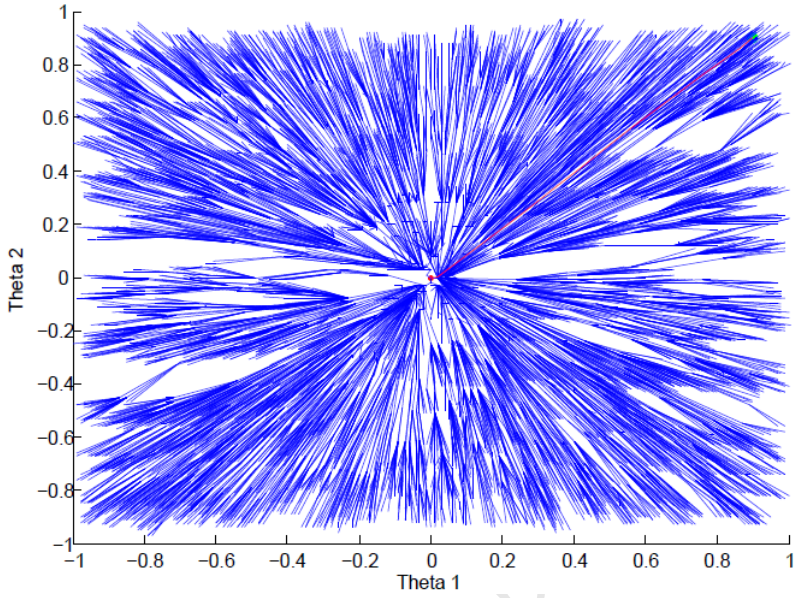


Figure 4.19: RRT* tree after 10000 iterations. Cost of the path is 1.2784 radians. CPU time is 1000 ms (1 second).

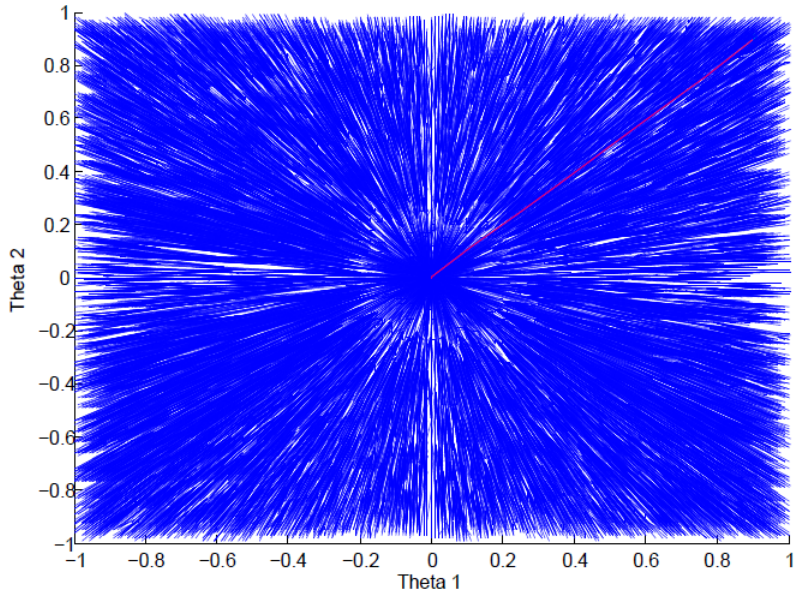


Figure 4.20: RRT* tree after 50000 iterations. Cost of the path is 1.2697 radians. CPU time is 14 000 ms (14 seconds).

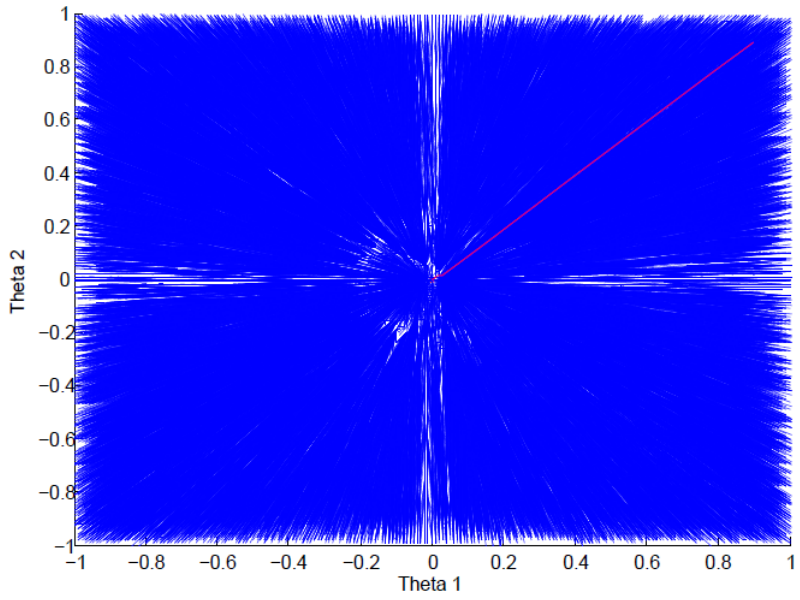


Figure 4.21: RRT* tree after 100000 iterations. Cost of the path is 1.2697 radians. CPU time is 73 000 ms (73 seconds).

In Figure 4.17, it took CPU time of < 0 ms for RRT* to produce a path with a cost of 1.2855 radians. This path is more optimal than those produced by RRT and RRT Ball. In Figure 4.18 for N = 5000, the cost of the path does not decrease much but still decrease to 1.2788 radians with a CPU time of < 0 ms, still better than that of RRT and RRT Ball in Figure 4.5 and 4.11. For $N = 10000$ in Figure 4.19, the cost of the path produced by RRT* seems to approach an optimal value with a cost value of 1.2784 radians and a CPU time of 1000 ms (1 second). An optimal path in this case would be a straight line joining the start point and the end point. Looking at Figure 4.19, the path is approaching a straight line. For N = 50 000 in Figure 4.17, it took CPU time of 14 seconds with a path cost 1.2697 radians which is the same path cost for N = 100 000 in Figure 4.21 with a CPU time of 73 seconds. The summary of the results of the RRT* algorithm in Figure 4.17 to 4.21 is given in Table 4.3.

The cost of the path produced by RRT* seems to decrease as the number of iterations increase. RRT* becomes computationally expensive when the number of iterations increase but an almost optimal path can be obtained without making this number big, i.e, about 5000 iterations or less. Based on these results, in a

Iterations	CPU time	Cost
2164	< 0.00	1.29
5000	< 0.00	1.28
10000	1.00	1.28
50000	14.00	1.27
100000	73.00	1.27

Table 4.3: Summary of the RRT* Algorithm. Time given in seconds and cost in radians.

2D case, RRT* outperforms RRT Ball and RRT in terms of the cost of the path produced with RRT Ball coming in second place. RRT* is able to find a shorter path quicker than RRT Ball while RRT finds a non-optimal path. This makes RRT* a good candidate for use in this project as a planning node but all three planners need to be tested on the Packbot manipulator before a final decision can be made.

The Packbot manipulator consists of 5 DOF that are being used in this project. This means the joint space in this case has 5 variables and each node has 5 joint variables, i.e, $q = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5)$. The DISTANCE function returns the 5D distance from one configuration to another, i.e,

$$Dist\_5D = \sqrt{(q_{new} - q_{old})^2} \tag{4.1}$$

$$Dist\_5D = \sqrt{(\theta_{1n} - \theta_{1o})^2 + (\theta_{2n} - \theta_{2o})^2 + (\theta_{3n} - \theta_{3o})^2 + (\theta_4 - \theta_{4o})^2 + (\theta_{5n} - \theta_{5o})^2} \tag{4.2}$$

where n represents the joints in the new node and o represents the joints in the old node. q_{goal} becomes $(\theta_{1goal}, \theta_{2goal}, \theta_{3goal}, \theta_{4goal}, \theta_{5goal})$ and q_{start} becomes $(\theta_{1start}, \theta_{2start}, \theta_{3start}, \theta_{4start}, \theta_{5start})$. The tree that is created is in 5D and the CreateBall function creates a ball in 5D which is difficult to visualize. The diagrams showing the trees created in a 2D case were for the purposes of understanding the algorithms and this is hard to show for a 5D case and no trees will be shown in a 5D case. The trajectory of the end-effector will be shown instead and simulation

results of the three planners in 5D are discussed and analyzed in Chapter 6.

Chapter 5

Tip-over Stability Measure

This chapter deals with the derivation of the Force-Angle stability measure model of the Packbot510i. Section 5.1 discusses examples of a Force Angle stability measure to understand how it is derived. The Force Angle stability measure was derived by Rey and Papadoupoulos in [5]. Section 5.2 discusses the derivation of the Force Angle stability measure model for the Packbot510i robot. This Force Angle stability measure model for Packbot510i was derived by Dube in [6].

5.1 Force Angle Stability Measure Examples

Below is a planar example as shown in [5]. Shown in Figure 5.1 is a two contact point planar system whose system Center of Mass (C.M) is subject to a net force $\mathbf{f_r}$ which is the sum of all forces acting on the mobile system except the supporting reaction forces (which do not contribute to a tip-over motion instability). This force vector subtends two angles, θ_1 and θ_2, with the two tip-over axis normals I_1 and I_2, and acts along a line which is at a distance of $||\mathbf{d_1}||$ and $||\mathbf{d_2}||$ respectively from the two tip-over axes. The Force-Angle stability measure, β, is given by the minimum of the product of θ_i, $||\mathbf{d_i}||$, and $||\mathbf{f_r}||$. Thus we have

$$\beta = \theta_i \cdot ||\mathbf{d_i}|| \cdot ||\mathbf{f_r}|| \tag{5.1}$$

Critical tip-over stability occurs when β, referred to as the stability index, goes to zero, i.e., when any θ_i becomes zero, or when any $||\mathbf{d_i}||$, or the force $\mathbf{f_r}$

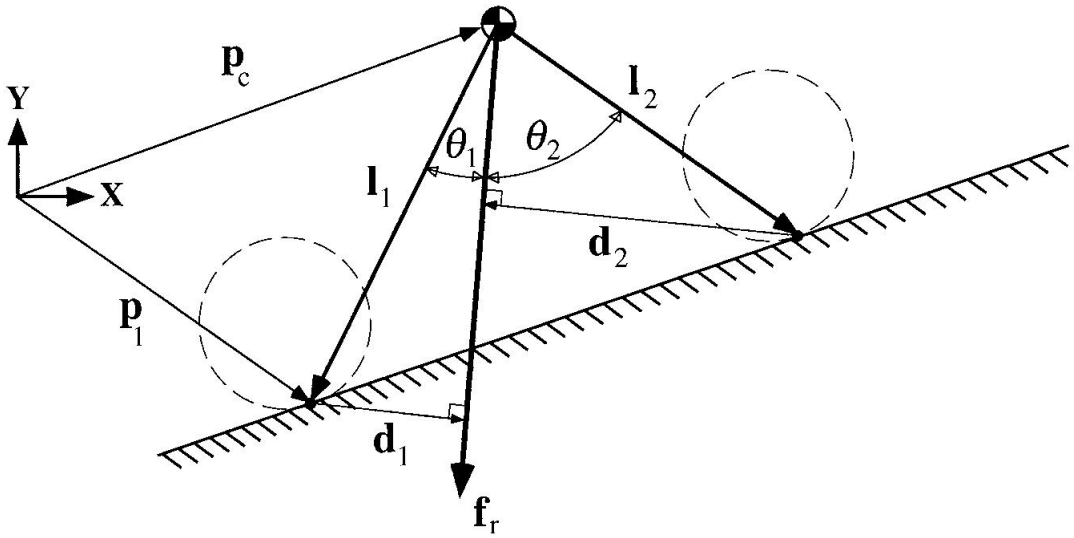


Figure 5.1: Planar Force-Angle stability measure -Planar [5]

become zero. Then angle θ_i becomes zero when the net force is coplanar with one of the tip-over axes $\mathbf{I_i}$, and this is the typical manner in which a tip-over instability occurs. If $\mathbf{f_r}$ lies outside the cone described by $\mathbf{I_1}$ and $\mathbf{I_2}$, the angle becomes negative and tip-over is in progress. The distance $||\mathbf{d_i}||$ becomes zero as the net force is coplanar with one of the tip-over axes, or, as the system C.M. approaches one of the tip-over axes. This latter case is less frequent, requiring a system with reconfigurable legged support or that a very large payload be held far below the support plane. Note that these two geometric parameters, θ_i and $||\mathbf{d_i}||$, together characterize the tip-over stability margin of the system. The angle θ_i captures the effect of top heaviness (where the term top heaviness is used to describe a change in the system C.M. height along the net force vector $\mathbf{f_r}$) and the distance $||\mathbf{d_i}||$ captures the effect of changes in the moment contribution of the net force. Finally, weighing by the magnitude of $\mathbf{f_r}$ is used to provide heaviness sensitivity since when $\mathbf{f_r}$ approaches zero, even the smallest disturbance may topple the vehicle.

For a mobile system which is capable of varying its C.M. height, or of carrying and manipulating a variable payload, it is important that the tip-over stability margin be sensitive to the reduced stability associated with an increase in the C.M. height along $\mathbf{f_r}$. For the Force-Angle stability, this is illustrated in Figure

5.2 where an increase in C.M. height clearly results in a smaller minimum angle and therefore, a reduced β. Note that measures which will use only a ratio of the normal forces at the ground contact points or a measure of the minimum moment exerted by $\mathbf{f_r}$ with respect to the tip-over axes, are not sensitive to stability margin changes associated with changes in the system C.M. height.

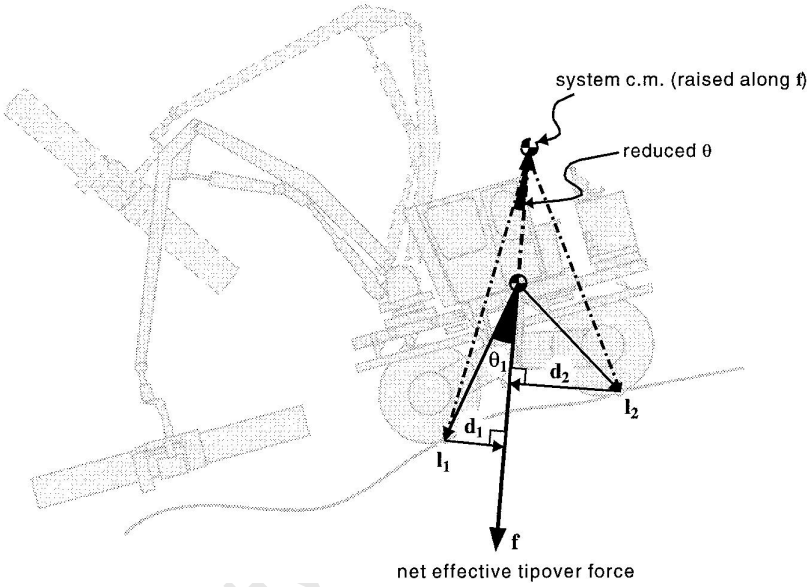


Figure 5.2: Effect of center-of-mass height [5]

For the general case, p_i represents the location of a ground contact point of the platform [5]:

$$p_i = (p_x, p_y, p_z)_i^T \tag{5.2}$$

and p_c represents the location of the system C.M.

$$p_c = \frac{\sum_j p_{mass_j} m_j}{m_{tot}} \tag{5.3}$$

where $i = 1, \cdots, n-1$ and P_{mass_j} is the location of the mass of the jth member of the system and m_{tot} is the total system mass.

The lines which join the ground contact points are the candidate tip-over mode axes, a_i.

$$a_i = p_{i+1} - p_i \tag{5.4}$$

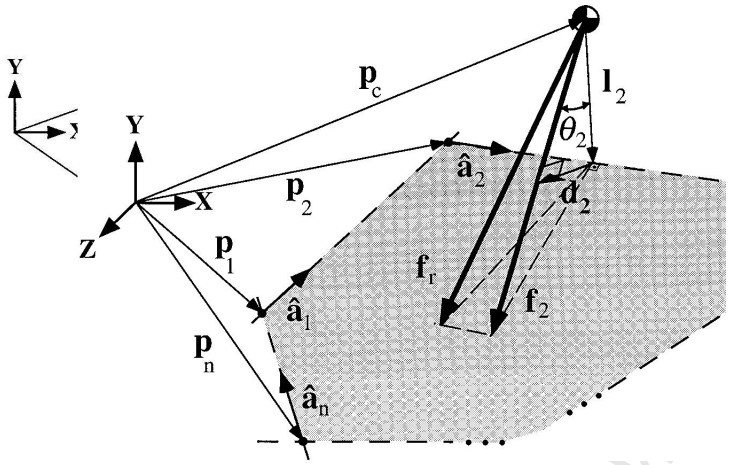


Figure 5.3: Force Angle Stability Measure - 3D [5]

$$a_n = p_l - p_n \tag{5.5}$$

$$\hat{a_i} = \frac{a}{||a||} \tag{5.6}$$

The tip-over axis normals l_i which pass through the system C.M. are:

$$l_i = (I - \hat{a_i}\hat{a_i}^T)(p_{i+1} - p_c) \tag{5.7}$$

where I is the 3×3 identity matrix.

The component of the force acting about each tip-over axis is:

$$f_i = (I - \hat{a_i}\hat{a_i}^T)n_r \tag{5.8}$$

The effective net force vector is:

$$f_i^* = f_i + \frac{\hat{l_i} \times n_i}{||\hat{l_i}||} \tag{5.9}$$

The distance of the force from the tip-over axes is:

$$d_i = -l_i + (l_i^T \cdot \hat{f_i}^*)\hat{f_i}^* \tag{5.10}$$

The angle of the force from the tip-over axes is:

$$\theta_i = \sigma_i \arccos(\hat{l}_i \cdot \hat{f}_i^{\,*}) \tag{5.11}$$

where:

$$0 \leq \frac{\theta_i}{\sigma_i} \leq \pi \tag{5.12}$$

and:

$$\sigma_i = \begin{cases} +1 & (\hat{f}_i^{\,*} \times \hat{l}_i) \cdot \hat{a} \\ -1 & otherwise \end{cases}$$

5.2 Force Angle Stability Model for Packbot510i

The modeling of the Packbot manipulator and flipper pose using the Force Angle stability measure is described below as shown in Dube [6]. Flippers can either be up or in contact with the ground as shown in Figure 5.4. The minimum and maximum angles, ϕ_{min} and ϕ_{max}, for which the flippers of the Packbot robot will be in contact with the ground are found as follows:

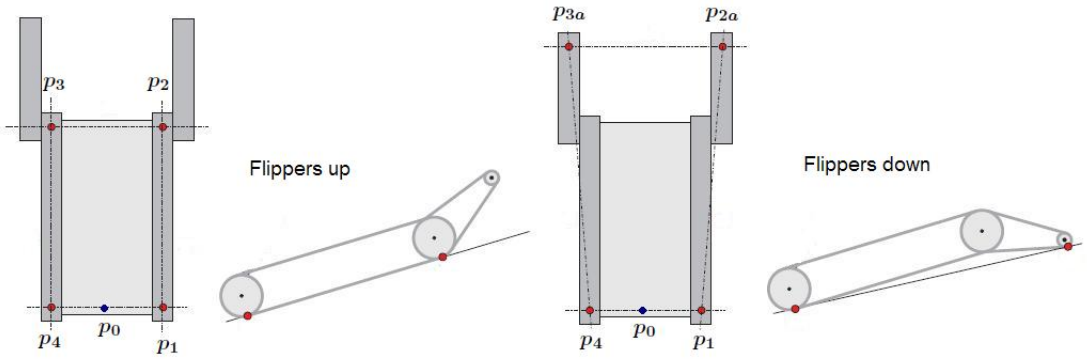


Figure 5.4: Flipper Positions [6]

$$\phi_{min} = \arcsin(\frac{H_{pb} - R_f}{L_f}) \tag{5.13}$$

$$\phi_{max} = 180 - \phi_{min} \tag{5.14}$$

where the Packbot dimensions are: L_{pb} is the length of the base, W_{pb} is the width of the base, H_{pb} is the height of the base.

Flippers are up when the flipper angle ϕ_f is outside the range of ϕ_{min} and ϕ_{max} i.e. $sin(\phi_f) > sin(\phi_{min})$. Given the position of the Packbot p_0, (here it is assumed the position is measured at the center of the rear axis of the Packbot), the slope vector s_p and normal of the slope n_p and the front of the Packbot w_p, the contact points are:

$$p_1 = p_0 - w_p \times \frac{W_{pb}}{2} \tag{5.15}$$

$$p_2 = p_1 + s_p \times L_{pb} \tag{5.16}$$

$$p_3 = p_2 + w_p \times W_{pb} \tag{5.17}$$

$$p_4 = p_3 - s_p \times L_{pb} \tag{5.18}$$

The vector and normal of the Packbot base are:

$$s_{pb} = s_p \tag{5.19}$$

$$n_{pb} = n_p \tag{5.20}$$

Once the flippers are in contact with the ground, i.e. if $sin(\phi_f) \leq sin(\phi_{min})$, the flipper motion changes the configuration of the Packbot. To calculate the configuration based on the flipper down angle, the contact points become, given the slope and normal of the slope and the front of the Packbot:

$$p_{2a} = p_1 + s_p \times L_3 \tag{5.21}$$

$$p_{3a} = p_{2a} + w_p \times W_{pb} \tag{5.22}$$

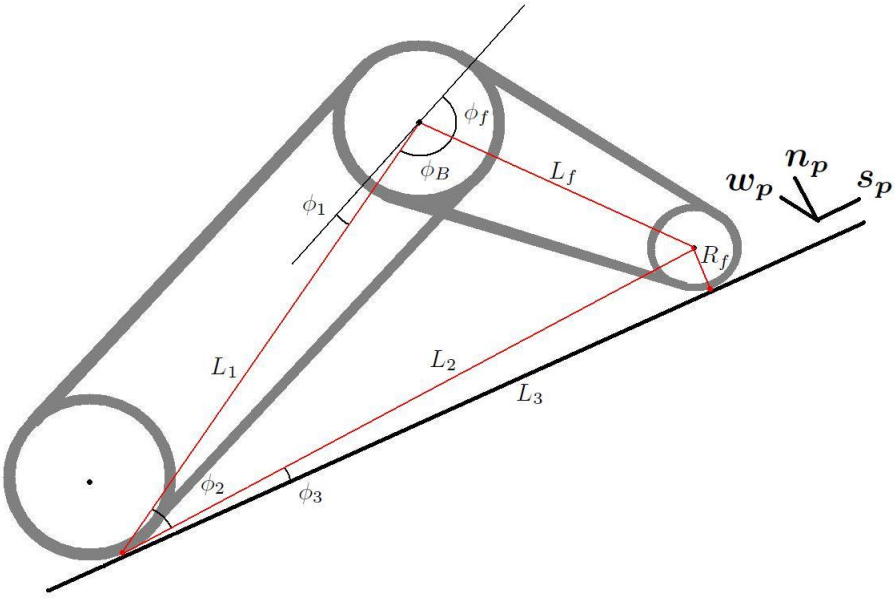


Figure 5.5: Flipper Contact [6]

$$p_2 = p_1 + s_p \times L_{pb} \times cos(\phi_2 + \phi_3 - \phi_1) + n_p \times L_{pb} \times sin(\phi_2 + \phi_3 - \phi_1) \quad (5.23)$$

where the lengths and angles as shown if Figure 5.5 are:

$$L_3 = \sqrt{L_2^2 - R_f^2} \quad (5.24)$$

$$L_2 = \sqrt{L_1^2 + L_f^2 - 2L_1 L_f \cos(\phi_B)} \quad (5.25)$$

$$L_1 = \sqrt{L_{pb}^2 + \left(\frac{H_{pb}}{2}\right)^2} \quad (5.26)$$

$$\phi_B = 180 - \phi_1 - \phi_f \quad (5.27)$$

$$\phi_1 = \arccos\left(\frac{L_{pb}}{L_1}\right) \quad (5.28)$$

$$\phi_2 = \arccos\left(\sqrt{\frac{L_1 + L_2 - L_f}{2L_1L_2}}\right) \tag{5.29}$$

$$\phi_3 = \arcsin\left(\frac{R_f}{L_2}\right) \tag{5.30}$$

The vector and normal of the Packbot base are:

$$s_{pb} = s_p \times \cos(\phi_2 + \phi_3 - \phi_1) + n_p \times \sin(\phi_2 + \phi_3 - \phi_1) \tag{5.31}$$

$$n_{pb} = s_{pb} \times w_p \tag{5.32}$$

Given the joint angles of the manipulator links, the position of the manipulator links are modeled using forward kinematics as done in Chapter 3. From the forward kinematics, the location q_j^* of the jth link is obtained. Transforming the manipulator link positions from the Packbot coordinate frame to the environment frame:

$$q_j = p_0 + s_{pb} \times q_m + n_{pb} \times H_{pb} + R \times q_j^* \tag{5.33}$$

where

$$R = [w_p, s_{pb}, n_{pb}] \tag{5.34}$$

q_m is the coordinate of the manipulator base in the Packbot frame.

The Packbot system center of mass is affected by the manipulator center of mass and the base center of mass. The center of mass position of the base is:

$$cm_b = p_1 + s_{pb} \times \frac{L_{pb}}{2} + w_{pb} \times \frac{W_{pb}}{2} + n_{pb} \times \frac{H_{pb}}{2} \tag{5.35}$$

The center of mass of each link cm_{qj} is placed at the center of the link. The center of mass of the manipulator is:

$$cm_m = \frac{\sum_j cm_{qj} \times m_{qj}}{m_m} \tag{5.36}$$

The center of mass of the Packbot system is thus:

$$cm_{pb} = \frac{cm_b \times m_{pb} + cm_m \times m_m}{m_b + m_m} \tag{5.37}$$

This method is based on the assumption that uneven terrains have a resulting slope that puts the system on a specific position and orientation that are input to the system.

Chapter 6

Simulation Results

Simulation results of the test performed on the selected methods are presented here. Tests were done on a Dell OptiPlex-760 running Linux Ubuntu 11.04(natty) with 3.2 GB RAM and an Intel(R) Core(TM)2 Duo CPU E8400 3.00 GHz. The software platform used is ROS and the Electric version was chosen. ROS stands for Robot Operating System. Initially, an introduction to ROS is provided in section 6.1. This is followed by simulations and analysis of kinematics modeling. Thereafter, the performance of the three planning algorithms introduced in Chapter 4 is evaluated, together with a discussion and motivation for the choice of the planning algorithm used in the safety inspections simulations. The three planning algorithms were coded and implemented on the Packbot manipulator kinematic model. The Force Angle stability measure model was implemented on the Packbot robot and the results are shown after this together with their discussions and analysis.

Simulation results of the complete system are then presented combining kinematics, planning and tip-over prevention. Finally, simulation results showcasing the performance of the mine safety inspections system are provided, together with a discussion of overall system limitations.

6.1 Introduction to Robot Operating Software (ROS)

ROS is an open source distributed robotics platform designed to accelerate robotics research and development, including commercial application development. It provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is a high-quality, actively maintained, well-documented software platform intended to support the academic and industrial robotics communities. ROS includes reusable components that implement a variety of low- and high-level functionality, such as base navigation, mapping, visual odometry, arm planning and control, data visualization, object recognition, and task-level execution. ROS supports a number of research robots and common robot simulators but not the Packbot robot. ROS is licensed under an open source, BSD license.

ROS supports a number of operating systems including Ubuntu, OS X, Fedora, OpenSUSE, Gentoo and Windows (See the ROS website for a list). A number of ROS versions have been released including ROS Turtle, Diamondback, Electric, Fuerte and ROS Groovy Galapagos with the latter being the latest. ROS Electric is used in this work because it was the latest at the time the project started off. There are tutorials available on the website that teach how to use ROS. See HTTP://www.ros.org/wiki/ROS/Tutorials for tutorials.

Code can be divided into separate nodes and these nodes can communicate with each via messages and services tools available in ROS. A message is a simple data structure, comprising typed fields. Standard primitives types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrary nested structures and arrays (much like C structs). Nodes communicate with each other by publishing messages to topics. One node publishes the message on a topic and the other node subscribes to this topic to receive the message. See http://www.ros.org/wiki/Messages for more information about ROS messages.

The publish/subscribe model is a very flexible communication paradigm, but its many-to-many one-way transport is not appropriate for RPC request/reply

interactions, which are often required in a distributed system. Request/reply is done via a service, which is defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Client libraries usually present this interaction to the programmer as if it were a remote procedure call. See http://www.ros.org/wiki/Services for more information about services. In this work, the end-effector goal position in Cartesian frame is sent to the Inverse Kinematics node via a service and the goal position in joint space is sent to the planning node via a service. The stability node is integrated into the Inverse Kinematics and Planning nodes because the computational time increases if the stability node is separate.

6.2 Kinematics

Simulation results for the kinematic modeling were obtained after implementing the kinematic model (Algorithm 1), derived in Chapter 3, on the Packbot manipulator. The Z - axis of the Packbot frame is pointing in the opposite direction of the global frame as shown in 6.1. The axes (X_G, Y_G, Z_G) are the axes of the global frame and (X_P, Y_P, Z_P) are the axes of the Packbot frame. This is due to the way the Packbot drivers have been written. This means when sending commands to the Packbot robot, all the joint angles need to be inverted. For the forward kinematics, the 5 joint angles were given and the transformation matrix T was computed using equations 3.12 to 3.23 as explained in section 3.2. The (x, y, z) position of the end-effector was extracted from this transformation matrix as $(x, y, z) = (r_{14}, r_{24}, r_{34})$ in the transformation matrix T from equation 3.11. The simulation was run 10 times for different joint angles and (x, y, z) positions were recorded. This simulation does not really verify the forward kinematic model derived but is used in developing the inverse kinematics. So if the inverse kinematics works the probability of the forward kinematics working is high. Experimental results will be given in Chapter 7 to show how closely the forward kinematic model represents the model of the robot. Table 6.1 shows a sample of joint angles configurations and their corresponding end-effector positions. The joint angles, θ_1 to θ_5 in Table 6.1 were plugged into Table 3.2 to

compute the DH parameters for the manipulator and the corresponding (x, y, z) values in Table 6.1 were computed using the forward kinematics model presented in section 3.2 as explained above. This simulation was run to test if the model is working, a complete test would include testing each and every configuration in the robot workspace but this is not practical. Therefore, only a sample of these configuration is given in Table 6.1.

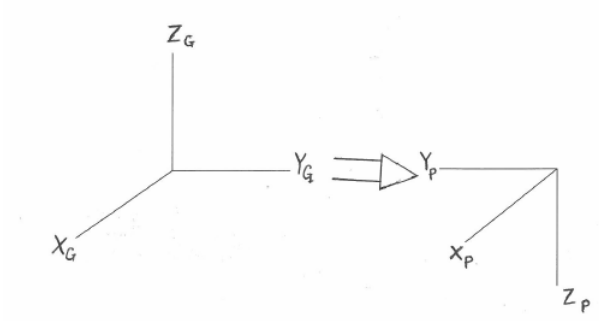


Figure 6.1: Transformation between the global frame and the Packbot frame.

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	0.00	-0.90	0.70	-0.20	0.50	1.92	-0.016	0.98
2	0.79	-0.30	0.90	-0.80	-0.20	1.53	1.26	0.12
3	-0.79	-2.00	1.90	-1.80	0.70	0.39	-0.17	1.43
4	1.42	-2.70	2.40	-0.80	0.30	0.33	0.41	1.22
5	-1.42	-2.70	2.40	-0.80	0.30	0.29	-0.41	1.22
6	2.36	-0.20	-2.40	1.80	-0.70	-0.07	0.35	1.14
7	-2.36	-0.20	-2.40	1.80	0.70	-0.16	-0.38	1.06
8	0.00	-0.20	-2.40	1.80	0.70	0.81	-0.016	1.05
9	0.00	-2.70	2.40	-1.80	0.10	-0.03	-0.016	1.22
10	0.00	-1.70	2.40	-1.80	0.10	0.99	-0.016	0.96

Table 6.1: Joint Angles With Their Corresponding End-Effector Positions

For the inverse kinematics, various (x, y, z) end-effector positions were given and their corresponding joint angles were computed. A sample of these joint angles for each end-effector position were then applied to the forward kinematics model derived in section 3.2 to verify that they correspond to the given end-effector position. Errors between the given end-effector positions and the calcu-

lated positions were recorded and analyzed. Table 6.2 to 6.6 show the inverse kinematics solutions for various end-effector positions and their corresponding forward kinematics solutions for highlighting errors between them.

The maximum absolute error in scenario 1 from Table 6.2 in all axes are $(x, y, z) = (0.01, 0.01, 0.01)$. In terms of percentages, the errors are $(2.00, 1.43, 1.00)$ and this means in all axes, the computed end-effector position would be off-set from the desired position by 1.31% which is the error resulting from the errors in all axes combined. In scenario 2 from Table 6.3 the maximum absolute errors in all axes are $(x, y, z) = (0.03, 0.03, 0.06)$ and in percentages they are $(6.00, 3.33, 5.00)$ and the end-effector would be off-set by 4.65%. In scenario 3 from Table 6.4 the maximum absolute errors in all axes are $(x, y, z) = (0.04, 0.02, 0.05)$ and in percentages they are $(4.44, 4.00, 6.25)$ and the end-effector would be off-set by 5.14%. In scenario 4 from Table 6.5 the maximum absolute errors in all axes are $(x, y, z) = (0.03, 0.03, 0.04)$ and in percentages they are $(4.28, 3.33, 5.00)$ and the end-effector would be off-set by 4.19%. In scenario 5 from Table 6.6 the maximum absolute errors in all axes are $(x, y, z) = (0.02, 0.02, 0.01)$ and in percentages they are $(2.22, 4.00, 1.11)$ and the end-effector would be off-set by 2.19%.

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	1.26	-0.12	-2.24	1.97	-0.74	0.50	0.71	1.00
2	1.26	-0.29	0.01	-2.14	0.97	0.49	0.70	1.00
3	1.26	-0.44	0.28	-2.26	1.11	0.49	0.70	1.01
4	1.26	-0.49	-1.83	2.29	-1.14	0.49	0.71	1.01
5	1.26	-1.19	1.89	-2.23	0.17	0.50	0.72	1.00
6	4.34	-2.78	2.00	-2.20	0.99	0.51	0.70	1.00
7	4.34	-2.73	-0.23	2.24	-1.12	0.51	0.70	1.01
8	4.34	-2.29	-1.09	2.39	-1.59	0.51	0.71	1.01
9	4.34	-1.79	-2.10	2.08	0.17	0.51	0.70	0.99
10	4.34	-1.86	-2.02	2.15	0.17	0.51	0.70	1.00

Table 6.2: Scenario 1 - End-Effector Position (0.5 0.7 1.0)

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	-0.90	-1.51	-1.10	-0.19	-1.10	-0.49	0.90	1.22
2	-0.90	-1.52	-1.07	-0.28	-1.12	-0.50	0.91	1.19
3	-0.90	-1.59	-0.88	-0.59	-1.23	-0.50	0.91	1.12
4	-0.90	-1.75	-0.54	-0.95	1.36	-0.50	0.91	1.26
5	-0.90	-1.90	-0.22	-1.19	1.43	-0.50	0.91	1.25
6	2.29	-1.63	1.10	0.19	1.10	-0.51	0.89	1.22
7	2.29	-1.62	1.07	0.28	1.12	-0.51	0.90	1.19
8	2.29	-1.27	1.49	-1.14	-0.17	-0.54	0.93	1.16
9	2.29	-1.21	0.18	1.22	-1.43	-0.51	0.90	1.24
10	2.29	-1.09	1.33	-1.35	-0.17	-0.53	0.92	1.18

Table 6.3: Scenario 2 - End-Effector Position (-0.5 0.9 1.2)

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	-0.43	-1.56	-1.38	-0.17	-1.45	-0.90	0.51	0.81
2	-0.43	-1.57	-1.35	-0.27	-1.48	-0.89	0.51	0.77
3	-0.43	-1.78	-0.88	-0.93	1.75	-0.92	0.52	0.85
4	-0.43	-1.87	-0.70	-1.11	1.82	-0.92	0.52	0.84
5	-0.43	-2.00	-0.46	-1.31	1.90	-0.92	0.52	0.84
6	2.75	-1.59	1.38	0.17	1.45	-0.91	0.49	0.81
7	2.75	-0.68	1.34	-1.76	-0.17	-0.93	0.50	0.78
8	2.75	-0.63	1.25	-1.79	-0.17	-0.93	0.50	0.78
9	2.75	-0.57	1.16	-1.81	-0.17	-0.92	0.50	0.79
10	2.75	-0.48	0.83	-1.83	1.71	-0.94	0.50	0.80

Table 6.4: Scenario 3 - End-Effector Position (-0.9 0.5 0.8)

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	0.74	-1.65	-1.29	-0.16	-1.45	-0.71	-0.89	0.81
2	0.74	-1.66	-1.26	-0.26	-1.47	-0.71	-0.89	0.78
3	0.74	-1.81	-0.91	-0.80	1.68	-0.73	-0.91	0.87
4	0.74	-1.98	-0.56	-1.13	1.81	-0.73	-0.91	0.84
5	0.74	-2.02	-0.48	-1.19	1.83	-0.73	-0.91	0.84
6	3.92	-1.49	1.29	0.16	1.45	-0.69	-0.91	0.81
7	3.92	-0.44	0.92	-1.73	-0.17	-0.70	-0.92	0.79
8	3.92	-0.45	0.94	-1.73	-0.17	-0.70	-0.92	0.79
9	3.92	-0.63	1.23	-1.67	-0.17	-0.71	-0.93	0.78
10	3.92	-0.47	0.76	-1.73	2.03	-0.69	-0.91	0.80

Table 6.5: Scenario 4 - End-Effector Position (-0.7 -0.9 0.8)

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	0.69	-0.05	-2.22	2.03	-0.92	0.90	0.51	0.90
2	0.69	-0.17	-2.11	2.15	-1.04	0.90	0.51	0.91
3	0.69	-0.26	-2.00	2.23	-1.14	0.90	0.51	0.91
4	0.69	-0.37	0.41	-2.31	0.98	0.90	0.51	0.91
5	0.69	-0.56	0.84	-2.40	0.81	0.90	0.51	0.91
6	3.76	-3.09	2.22	-2.03	0.92	0.91	0.49	0.90
7	3.76	-2.88	2.00	-2.23	1.14	0.91	0.49	0.91
8	3.76	-2.35	-1.40	2.40	-0.44	0.92	0.50	0.90
9	3.76	-2.23	-1.64	2.35	0.19	0.89	0.48	0.90
10	3.76	-1.91	-2.12	2.10	0.17	0.92	0.50	0.89

Table 6.6: Scenario 5 - End-Effector Position (0.9 0.5 0.9)

Forward kinematics consists of many matrix multiplication stages using OpenCV2 matrix library. This often results in computational errors such as rounding off values. Now, the forward kinematics is used when calculating the inverse kinematics solution as done in section 3.2 when calculating θ_3 (page 40). Since the position of (x_2, y_2, z_2) in Figure 3.10 in section 3.2 has some computational errors associated with it, the rest of the computation will be affected by these errors. Also, the forward kinematics model is applied to the computed joint angles to verify that they correspond to the desired end-effector position. Therefore, these errors in end-effector positions discussed above are a result of the computational errors when calculating (x_2, y_2, z_2) and the verification process. Since these errors are small, the kinematic model derived can be used in the mining safety inspections.

6.3 Planning

In this section, simulation results of the three RRT planning algorithms discussed in Chapter 4 are presented. The three algorithms were coded and implemented on the Packbot manipulator kinematic model. A random goal configuration was given to all three algorithms and their respective end-effector trajectories are plotted. A random goal was used as a way of testing the algorithms because using all the points in the workspace is impractical. The time it took for each planner to find a path as well as the cost of the path in joint space and workspace were recorded for analysis purposes. The path cost in workspace is the total euclidean distance traveled by the end-effector along a given path in meters and the path cost in joint space is the total distance traveled by all the joints along the path in joint space as shown by equation 4.1 and 4.2 in section 4.3. Unfortunately, it is difficult to display the trees of the three planners in 5D.

The first experiment was run for a goal position of $(x, y, z) = (0.20, 0.30, 1.00)$ which has many inverse kinematics solutions. The choice of the goal position is random as mentioned above. One solution was chosen as the goal node for planning, which is $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5) = (-1.45, -2.04, 1.74, -2.79, 1.09)$ and the start node is the home configuration of the robot, which is $(0.0, -2.88, 2.76, -2.97, 0.0)$. This particular joint space goal position was selected because it is one of the few

solutions that produce a shorter path which is good for illustration purposes. Any of the few solutions that produce a shorter path could have been used. The same experiment was run 15 times for each planner and the CPU times it took to plan the path are shown in Table 6.7.

Num	RRT	RRT*	RRT BALL
1	0	0	178340
2	0	0	176780
3	10	1000	178690
4	0	1000	175220
5	0	0	180440
6	10	1000	175890
7	0	1000	176530
8	0	1000	175500
9	10	1000	179610
10	0	0	183390
11	10	0	178620
12	0	0	178690
13	10	1000	177620
14	10	1000	178380
15	0	1000	176330

Table 6.7: CPU Computation times in milliseconds for RRT, RRT* and RRT BALL up to 5000 iterations.

Figure 6.2 and Figure 6.3 show the end-effector trajectories planned using the RRT planner from two different views. Figure 6.4 and Figure 6.5 show the end-effector trajectories planned using the RRT* from two different views and Figure 6.6 and Figure 6.7 show the end-effector trajectories planned using RRT BALL.

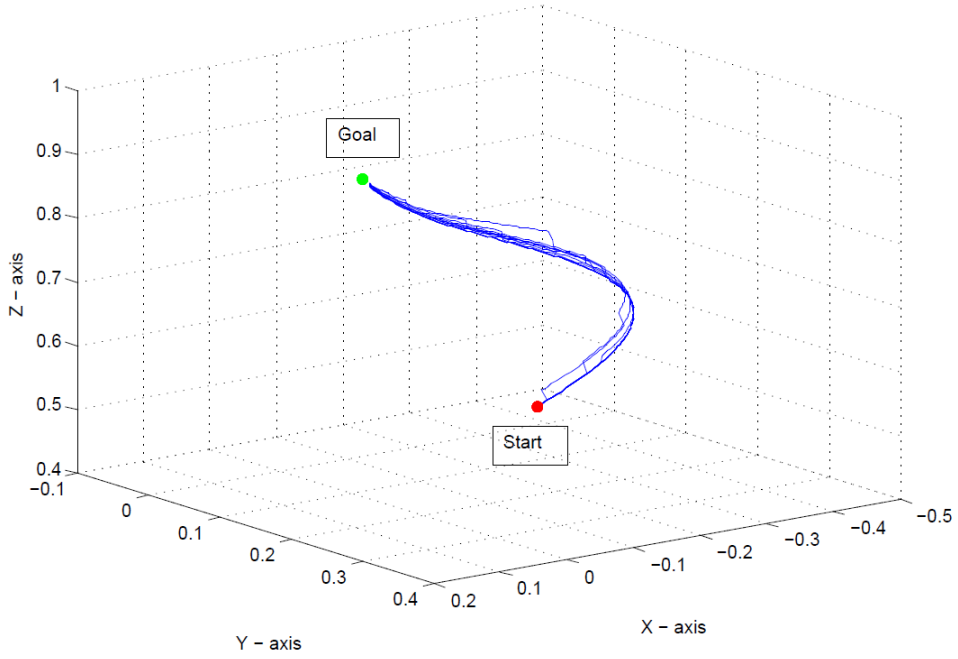


Figure 6.2: RRT End-effector trajectories (View 1). Ran 15 times for N = 5000 iterations.

RRT plans a path randomly without incorporating the notion of path cost. Looking at Figure 6.2 and 6.3, it produces 15 different paths to the same goal randomly. This is expected of it because of its random nature. An exact planner would probably produce the same path all the time given the same conditions. The geometry of the path has to do with the structure of the manipulator and how the planner is configured to produce the path. In this case the RRT planner has been configured to randomly sample all the joint angles in the configuration space at the same rate. In fact, all three planners have been configured to have this behavior. This means they do not not keep some joint angles constant while sampling the rest which would result in a different geometry of the path. This would result in a better or worse trajectory but is not dealt with in this thesis and is part of future work.

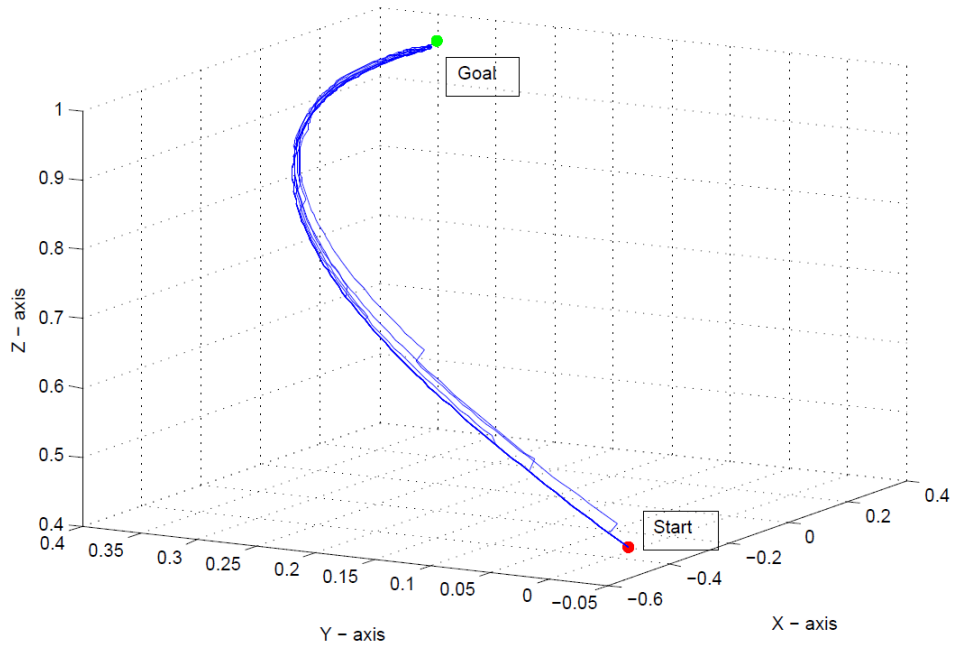


Figure 6.3: RRT End-effector trajectories (View 2). Ran 15 times for N = 5000 iterations.

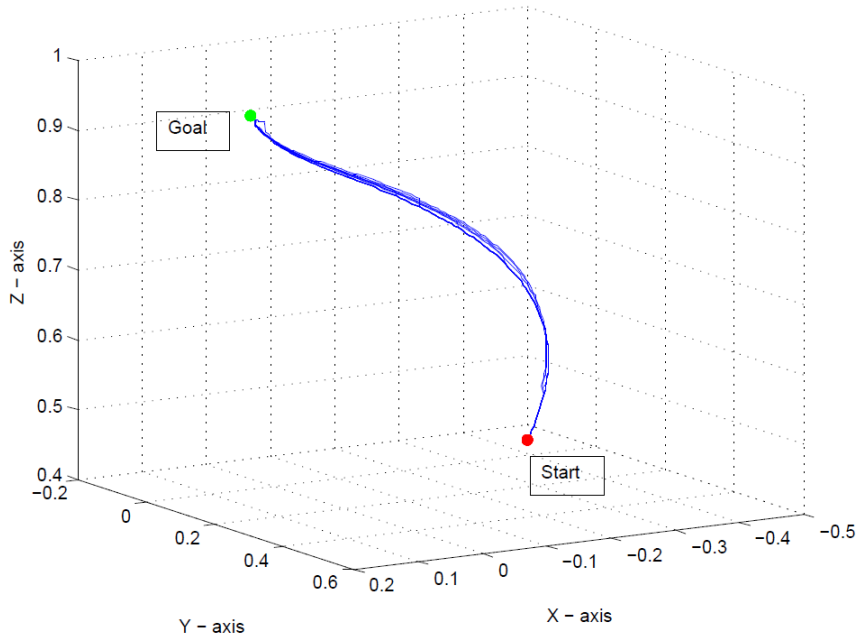


Figure 6.4: RRT* End-effector trajectories (View 1). Ran 15 times for N = 5000 iterations.

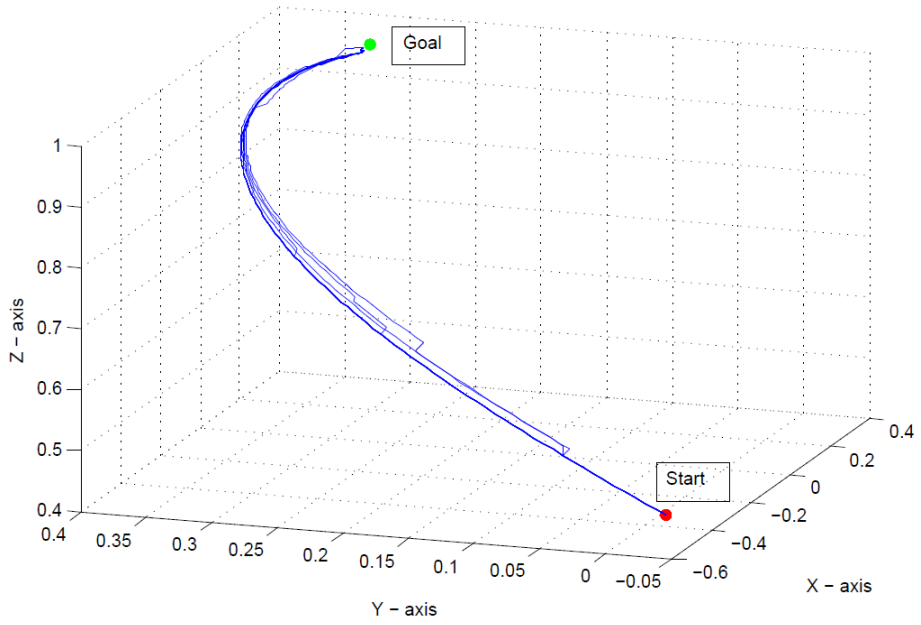


Figure 6.5: RRT* End-effector trajectories (View 2). Ran 15 times for N = 5000 iterations.

RRT* plans a path randomly taking the cost of the path in joint space into account, i.e, it tries to minimize the joint movements which as a result minimizes the path traversed by the end-effector. As shown in Figure 6.4 and 6.5, the variation in the 15 paths is smaller compared to the trajectories produced by RRT. This is expected since it tries to select a path with the minimum cost all the time. All the paths with the costs approaching the minimum (optimal) value would have similar geometries, for example, in a 2D case in Chapter 3, all the path tend to approach a straight line path as it is an optimal path with no obstacles. A small variation is observed and this is because the RRT* although perceived to be more optimal than RRT, is still a random planner and will not produce the same path all the time.

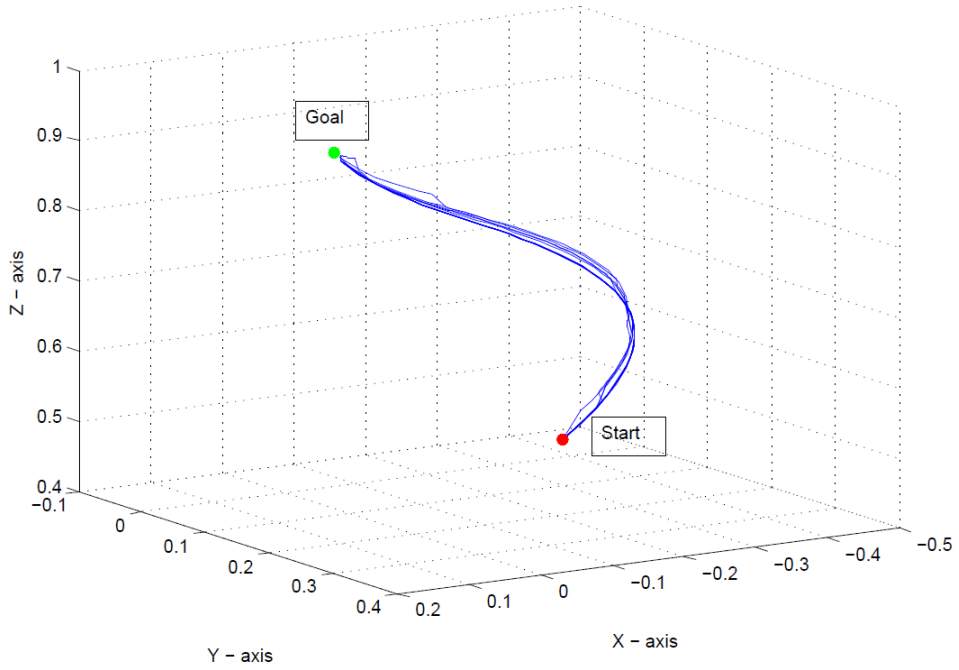


Figure 6.6: RRT BALL End-effector trajectories (View 1). Ran 15 times for N = 5000 iterations.

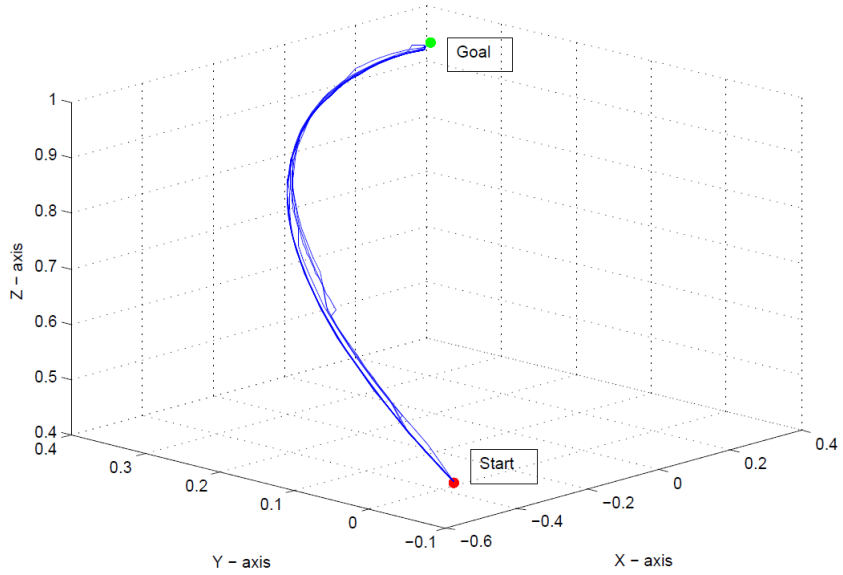


Figure 6.7: RRT BALL End-effector trajectories (View 2). Ran 15 times for N = 5000 iterations.

RRT Ball also plans a path taking the cost of the path in joint space into account. From Figure 6.6 and 6.7, RRT Ball seems to have more variation in the trajectories compared to RRT* but still less compared to RRT. A planner that produces trajectories with a small variations is preferred as it proves to be more reliable than others. The RRT* seems to be more reliable in this case but just looking at these trajectories is not sufficient to make a choice. In terms of computational intensity, the RRT Ball takes on average about 178 seconds to plan a path which is impractical. The RRT ranges from 0 to 10 milliseconds with RRT* ranging from 0 to 1 second.

Table 6.8 and 6.9 show the path costs of the 15 trials in the workspace as well as joint space for the three planners. These costs are also highlighted in Figure 6.8 and 6.9.

Num	RRT	RRT*	RRT BALL
1	1.02	1.02	1.01
2	1.02	1.02	1.02
3	1.02	1.02	1.01
4	1.05	1.03	1.01
5	1.03	1.02	1.01
6	1.02	1.02	1.04
7	1.03	1.02	1.01
8	1.03	1.02	1.01
9	1.02	1.01	1.01
10	1.02	1.01	1.01
11	1.02	1.01	1.01
12	1.02	1.01	1.01
13	1.03	1.02	1.01
14	1.03	1.01	1.01
15	1.02	1.04	1.01

Table 6.8: End-effector path costs in workspace in meters for RRT, RRT* and RRT BALL up to 5000 iterations.

The shortest path cost in workspace from Table 6.8 for RRT is 1.02 meters, the average path cost is 1.025 meters with a standard deviation of 0.0810 and the maximum path cost is 1.05 meters. The shortest path cost for RRT* is 1.01 meters, the average path cost is 1.019 meters with a standard deviation of 0.0068

and the maximum path cost is 1.04 meters. The shortest path cost for RRT BALL is 1.01 meters, the average path cost is 1.0127 meters with a standard deviation of 0.0075 and the maximum path cost is 1.04 meters.

Num	RRT	RRT*	RRT BALL
1	2.31	2.32	2.30
2	2.29	2.30	2.31
3	2.32	2.30	2.29
4	2.38	2.33	2.31
5	2.38	2.33	2.29
6	2.29	2.32	2.38
7	2.37	2.31	2.29
8	2.37	2.30	2.29
9	2.31	2.32	2.29
10	2.32	2.30	2.29
11	2.29	2.30	2.29
12	2.32	2.30	2.29
13	2.38	2.32	2.29
14	2.39	2.30	2.29
15	2.29	2.38	2.29

Table 6.9: Path costs in joint space in radians for RRT, RRT* and RRT BALL up to 5000 iterations.

The shortest path cost in joint space from Table 6.9 for RRT is 2.29 radians, the average path cost is 2.34 radians with a standard deviation of 0.035 and the maximum path cost is 2.39 meters. The shortest path cost for RRT* is 2.30 radians, the average path cost is 2.32 radians with a standard deviation of 0.020 and the maximum path cost is 2.38 radians. The shortest path cost for RRT BALL is 2.29, the average path cost is 2.30 meters with a standard deviation of 0.022 and the maximum path cost is 2.38 radians.

In terms of the path cost in workspace and in joint space, all three planners produce almost the same results, as shown in Figures 6.8 and 6.9, with RRT* having the least standard deviation in both the workspace and the joint space. The distances traveled in all three planners in workspace and joint space as shown in Figures 6.8 and 6.9.

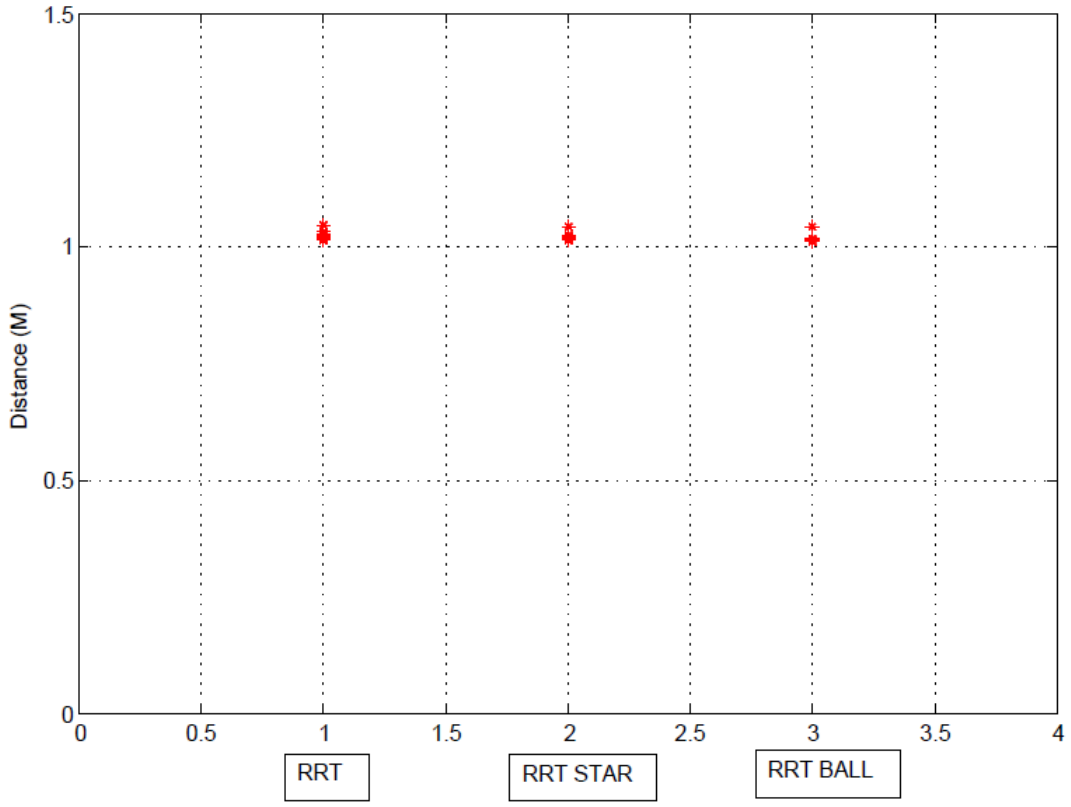


Figure 6.8: Workspace Path costs. Ran 15 times for N = 5000 iterations. The Y - axis represents the end-effector distance in meters and the red dots are the distances for the 15 trials.

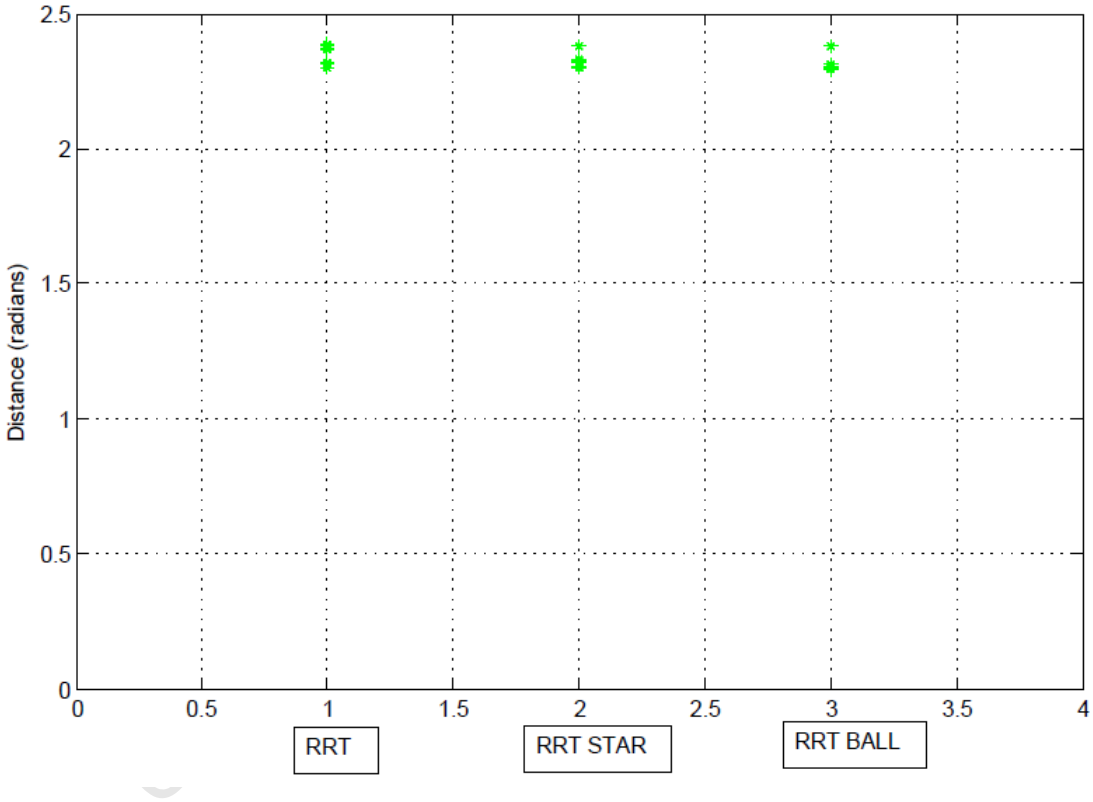


Figure 6.9: Joint Space Path costs. Ran 15 times for N = 5000 iterations. The Y - axis represents the 5D joints distance in radians and the green dots are the distances for the 15 trials.

The difference between the average path cost of RRT* and RRT in workspace is about 6 mm and the difference between RRT and RRT Ball in workspace is 12.3 mm. These differences in path cost are very small which means in this case both the RRT* and RRT Ball are not providing any significant improvement over RRT. In terms of the average path cost in joint space, the difference in average cost between RRT* and RRT is 0.02 radians and the difference between RRT Ball and RRT is 0.04 radians. Again, this is not a significant improvement over RRT for both RRT* and RRT Ball. Tables 6.10 and 6.11 summarize the comparison of the three planners in terms of path cost in workspace and joint space.

Num	RRT	RRT*	RRT BALL
Min	1.020	1.010	1.010
Mean	1.025	1.019	1.013
Max	1.050	1.040	1.040
Std	0.081	0.007	0.008

Table 6.10: Comparison of the three planners in Workspace. Values have been rounded off to keep the same number of significant figures.

Num	RRT	RRT*	RRT BALL
Min	2.290	2.300	2.290
Mean	2.340	2.320	2.300
Max	2.390	2.380	2.380
Std	0.035	0.020	0.022

Table 6.11: Comparison of the three planners in Joint Space. Values have been rounded off to keep the same number of significant figures.

Simulation results presented in this section suggest that all three planners perform similarly in terms of path cost in workspace and joint space. The RRT* and RRT Ball produce trajectories which are on average 6 mm and 12.3 mm shorter than RRT in workspace, respectively. In joint space RRT* and RRT Ball produce trajectories that are 0.02 radians and 0.04 radians shorter than RRT, respectively. This shows that RRT* and RRT Ball are not offering any significant improvement over RRT. However, the computational burden associated with RRT Ball makes it impractical to be used in the safety inspection system. RRT* is

sometimes slower compared RRT but can be implemented on a practical system. RRT* has a lower standard deviation in its trajectories which suggests that it is more reliable compared to RRT. Since RRT Ball proved to be impractical, it is not suitable to be used as a planner in this project and only RRT and RRT* will be compared when a stability measure is introduced in order to find a suitable planner.

6.4 Tip-over Stability Measure

The stability index used in tip-over prevention, defined in Chapter 5 equation 5.1, describes how likely a certain arm configuration is to overturn the robot system. Index of 1 represents the home configuration of the robot arm where the center of mass lies close to the center of the supporting polygon. This configuration is considered the most stable, however, there might be more stable configurations whose stability indices are greater than 1. Stability indices greater or equal to 0 are considered stable with 0 representing the less stable configurations and 1 or greater representing the most stable configurations. All indices less than 0 are considered unstable.

The stability measure needs to be tested on a real system to determine how accurate it detects the tip-over point of the system. This will be done in the next chapter. In this section, stability indices of random configurations (stable and unstable configurations) are computed and tabulated in Table 6.12. This is done to show the stability index varies for configurations close to and away from the home configuration. These stability indices will be used in the next section when simulating the complete system integrating all the components.

Arm configuration number one in Table 6.12 is the home configuration, see Figure 6.10, and its stability index is 1.02 as expected. As the arm moves away from the home configuration its stability index decreases as can been in row two to six of Table 6.12 with the configuration in row six shown in Figure 6.4. This stability index decreases until the system comes to point of tip-over in row 7 shown in Figure 6.4 and eventually tips over as can be seen in the configurations in row eight and nine with nine shown in Figure 6.13. When the arm is fully stretched as in configuration row 10, see Figure 6.14, the system becomes very

Num	θ_1	θ_2	θ_3	θ_4	θ_5	Index
1	0.00	-2.88	2.76	-2.97	0.00	1.02
2	0.00	-2.30	2.76	-2.97	0.00	0.83
3	4.38	-2.50	2.10	-2.49	0.89	0.19
4	1.31	-0.21	-2.52	2.09	-0.42	0.15
5	-0.69	-2.34	1.49	-2.28	1.39	0.31
6	-0.69	-2.34	-1.20	2.30	-0.58	0.09
7	2.51	-0.44	0.40	-2.20	0.94	0.00
8	-3.77	-3.05	0.28	1.90	-0.96	-0.29
9	0.00	-0.57	0.00	0.0 0	0.00	-1.29
10	0.00	0.00	0.00	0.00	0.00	-3.46

Table 6.12: Stability indices for 10 different manipulator configurations.

unstable as indicated by the stability index of -3.46.

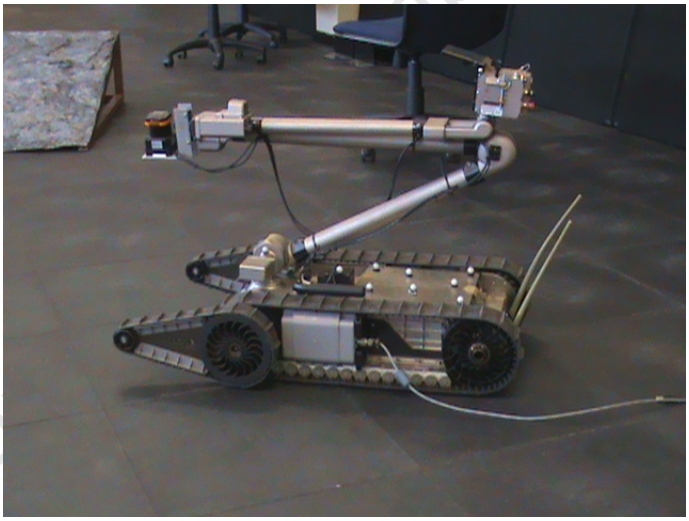
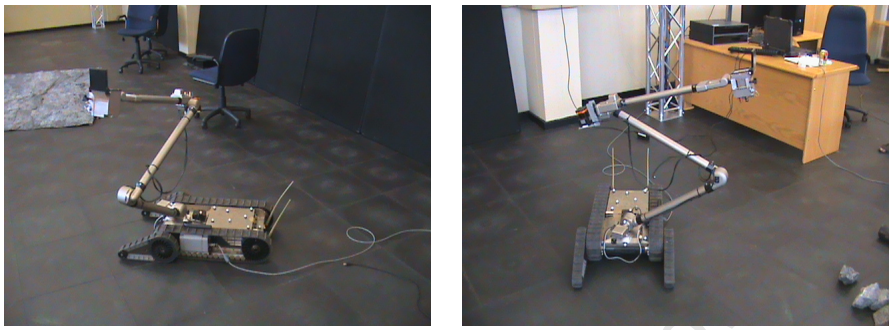
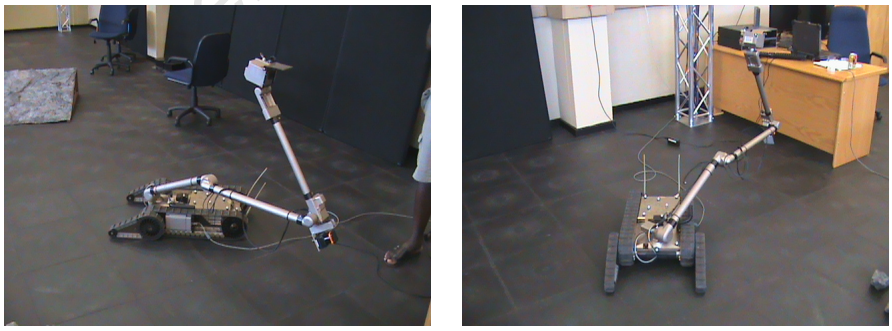


Figure 6.10: Home Configuration with stability index of 1.02.



(a) Side View (b) Front View

Figure 6.11: Configuration with joint positions (4.38, -2.50, 2.10, -2.49,0.00) and stability index of 0.19.



(a) Side View (b) Front View

Figure 6.12: Configuration with joint positions (2.51, -0.44, 0.40, -2.20,0.94) and stability index of 0.00.

Figure 6.13: Configuration with joint positions (0.00, -0.57 0.00 0.00 0.00) and stability index of -1.29.

Figure 6.14: Configuration with joint positions (0.00, 0.00, 0.00, 0.00, 0.00) and stability index of -3.46.

Simulation results presented in this section indicate that the Force Angle stability measure is able to predict point of tip-over and adjust the stability index accordingly. Configurations that keep the system stable are assigned a positive stability index and this stability index decreases as the system becomes unstable. Configurations that make the system unstable are assigned a negative stability index accordingly. This shows that the Force Angle stability measure can be used in tip-over prevention during the mine safety inspection process.

6.5 Combined System

Now all three components, kinematics, planning and tip-over stability, are integrated together to form a complete system. Several simulation experiments were run to test the combined system. It is clear from Table 6.7 of Section 6.3 that the RRT BALL is impractical as it takes at best 175.22 seconds CPU time to find a path, almost 175 times slower than RRT* and 17522 times slower than RRT. Therefore, only RRT and RRT* were used as planning nodes in this section and their results were analyzed. In these simulation experiments, a goal position was given in Cartesian frame and the corresponding inverse kinematics solutions were found in joint space. Firstly, an unstable inverse kinematics solution was chosen as a goal node to be sent to the planning nodes. A path was produced by planning nodes without taking system stability into account, this was done to highlight how unstable the system can be without avoiding this tip-over instability. Secondly, a stable inverse kinematics solution was chosen and sent as a goal node. A path was produced again without taking system stability into account. Thirdly, a stable solution was chosen as a goal node and a path was produced taking system stability into account. Finally, an unstable inverse kinematics solution was chosen as a goal node to see if the planners would be able to find a stable path to an unstable goal node.

In the first simulation experiment, the workspace goal position was $(x, y, z) = (0.40, 0.60, 1.00)$ and an unstable joint space goal position, from the inverse kinematics node, $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5) = (1.36, -0.18, -0.39, -2.06, 1.34)$ with a stability index of -0.11 was chosen. These goals were chosen randomly for illustration purposes and any goal in the workspace could have been used and any

inverse kinematics solution in the list of unstable joint space goals could have been used. The two planners RRT and RRT* planned a path without considering stability in this case, from the home configuration $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5) = (0.00, -2.70, 2.88, -2.69, 0.00)$. The planners were run for $N = 5000$ iterations and were not terminated when the goal was found. Figure 6.15 and Figure 6.16 show the end-effector trajectories from both planners and Figure 6.17 shows the stability indices of each node in the path from both planners.

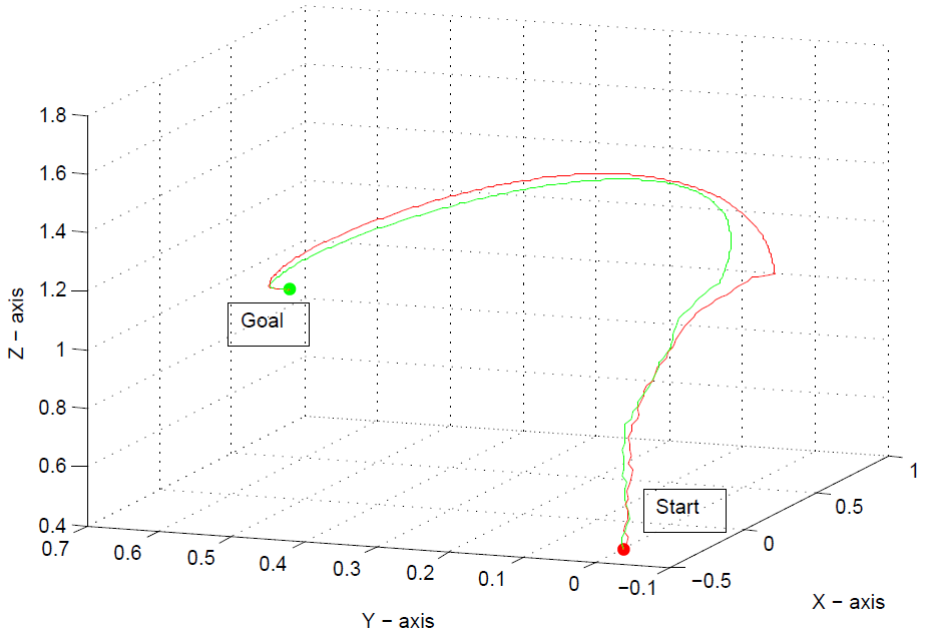


Figure 6.15: End-Effector trajectories for RRT and RRT* without Stability (View 1).

The RRT* workspace distance is 2.42 meters and the RRT workspace distance is 2.27 meters. The joint space distance of RRT* is 5.41 radians and for RRT is 5.24 radians. The RRT* workspace distance is 0.15 meters (15 cm) longer than the RRT workspace distance and the joint space distance is 0.17 radians longer than RRT. RRT* took CPU time of < 0 ms and RRT took CPU time of 990 ms. In this case RRT achieved a shorter distance at the cost of taking longer time to plan a path compared to RRT*. The stability indices of the nodes in the paths produced by the two planners went below zero at some point. This means the system became unstable as the manipulator traversed these paths.

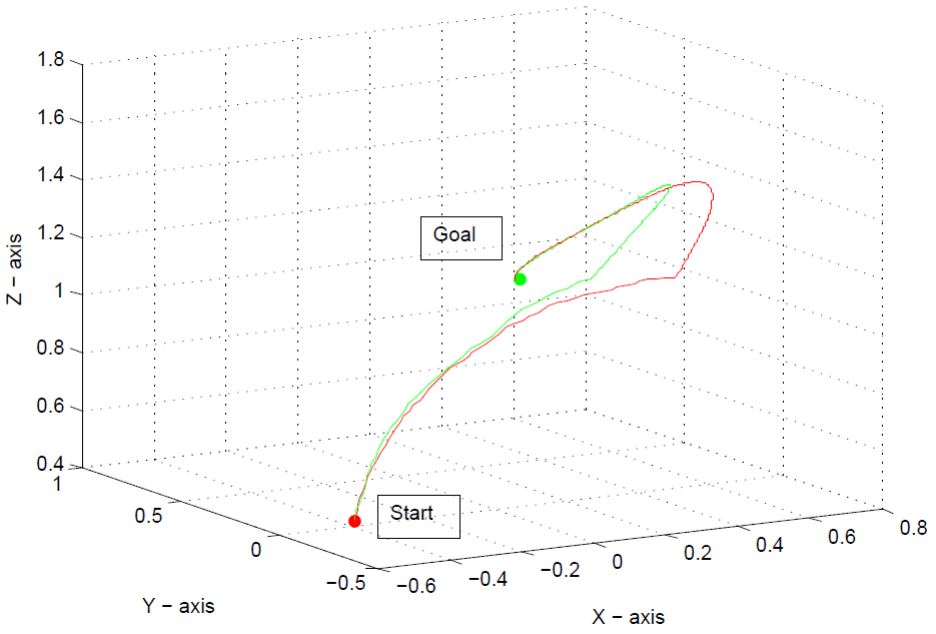


Figure 6.16: End-Effector trajectories for RRT and RRT* without Stability (View 2).

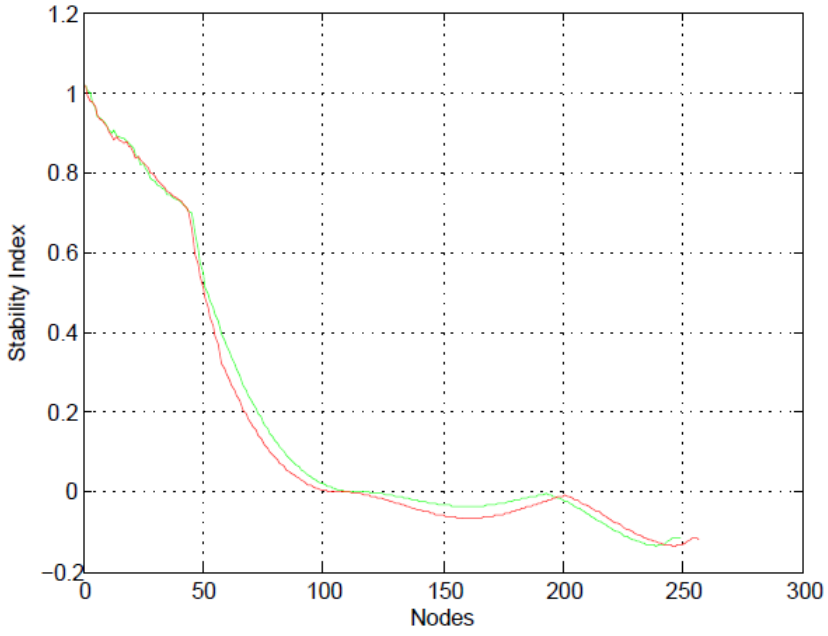


Figure 6.17: Stability Indices for RRT and RRT* without Stability. Green represents RRT and red represents RRT*.

This is expected since the final configuration of the arm is unstable and no stability checks were done in the planning stages.

In the second simulation experiment the same workspace goal was used and a stable joint space goal of $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5) = (1.36, -1.46, 2.24, -2.20, -0.17)$ with a stability index of 0.14 was chosen. The two planners planned a path without considering stability and ran for $(N = 5000)$ iterations and not terminated when the goal was found. Figure 6.18 and Figure 6.19 show the end-effector trajectories from both planners and Figure 6.20 shows the stability indices of each node in the path from both planners.

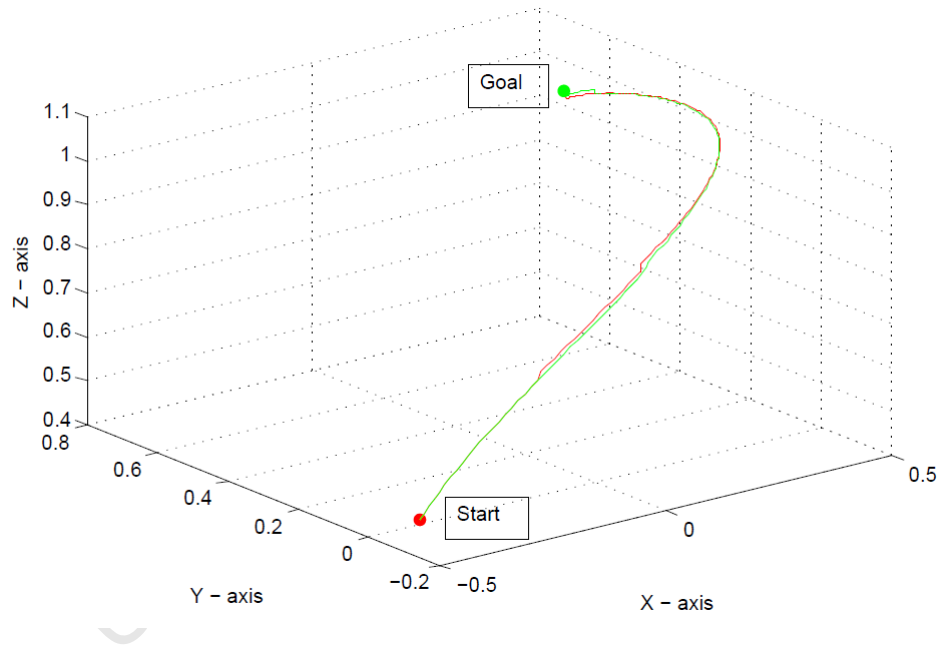


Figure 6.18: End-Effector trajectories for RRT and RRT* without Stability (View 1).

In simulation experiment number two, the workspace distance produced by RRT* was 1.69 meters and for RRT was 1.70 meters. The joint space distance produced by RRT* was 2.26 radians and for RRT was 2.27 radians. The workspace distance for RRT* was 0.01 meters shorter than RRT and the joint space distance was 0.01 radians shorter than RRT. RRT* took CPU time of < 0 ms to find a path and RRT took CPU time of 240 ms for RRT. The distance of the path produced by RRT was not significantly shorter than that of RRT (about 10 mm shorter).

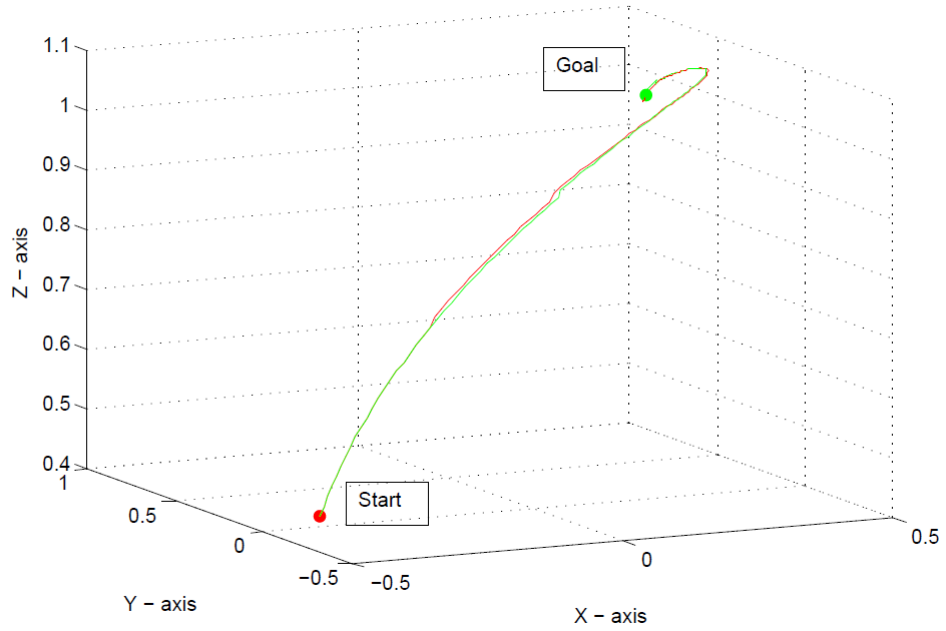


Figure 6.19: End-Effector trajectories for RRT and RRT* without Stability (View 2).

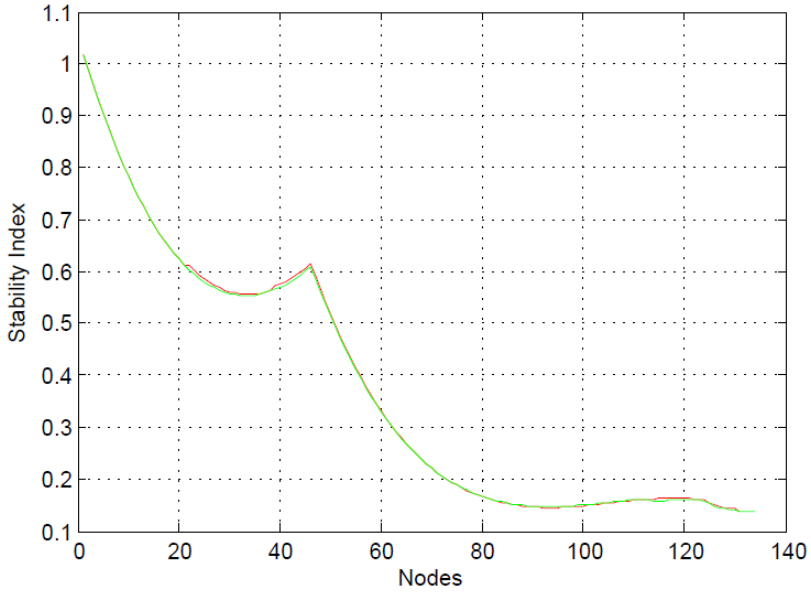


Figure 6.20: Stability Indices for RRT and RRT* without Stability. Green represents RRT and red represents RRT*.

The stability indices stayed above zero for both planners which means the system was stable when traversing the paths as can be seen in Figure 6.20. One thing to note here in these two experiments is that a stable goal resulted in shorter paths for both planners (1.69 compared to 2.42 for RRT* and 1.70 compared to 2.27 for RRT). The reason is that the arm is most likely to be extended away from the base resulting in a longer path to get there for unstable configurations. This might not always be the case and would need a lot of experiments to verify.

In the third simulation experiment the same workspace goal and joint space goal were used as the previous ones. The two planners planned a path considering stability and ran for ($N = 5000$) iterations and not terminated when the goal was found. Figure 6.21 and Figure 6.22 show the end-effector trajectories from both planners and Figure 6.23 shows the stability indices of each node in the path from both planners.

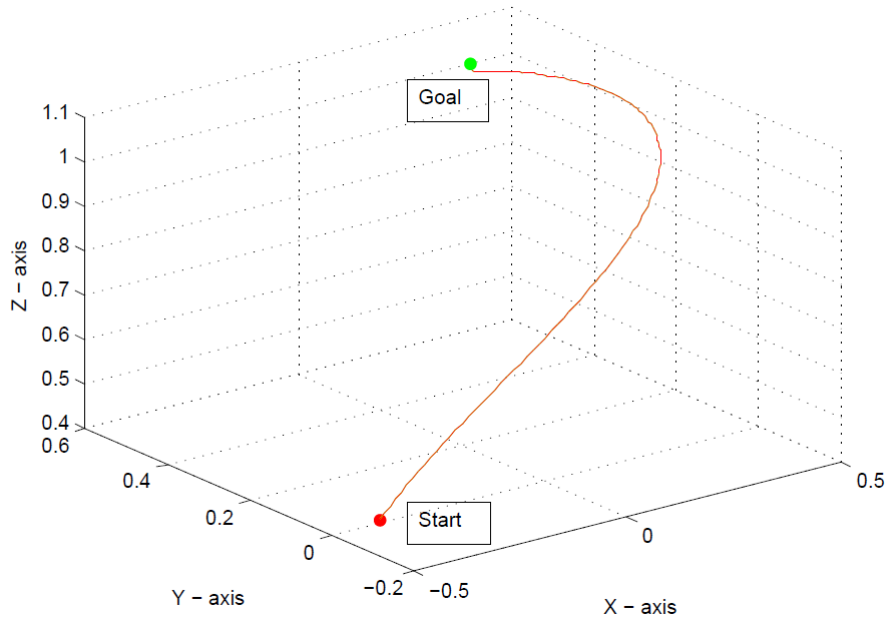


Figure 6.21: End-Effector trajectories for RRT and RRT* with Stability (View 1).

In this case both planners produced the same path length both in workspace and joint space of 1.69 meters and 2.23 radians respectively. RRT* took CPU time of 2 seconds and RRT took CPU time of 1.74 seconds.

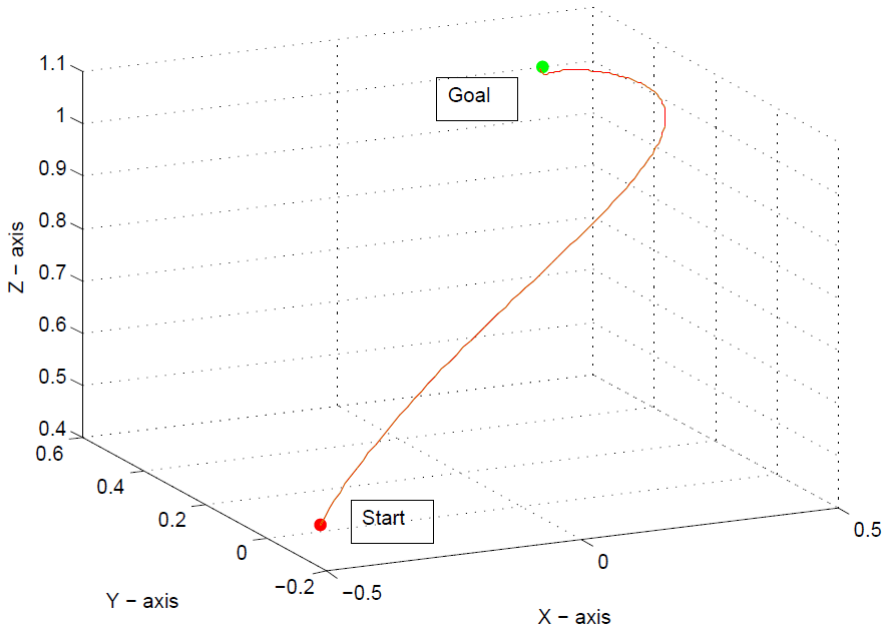


Figure 6.22: End-Effector trajectories for RRT and RRT* with Stability (View 2).

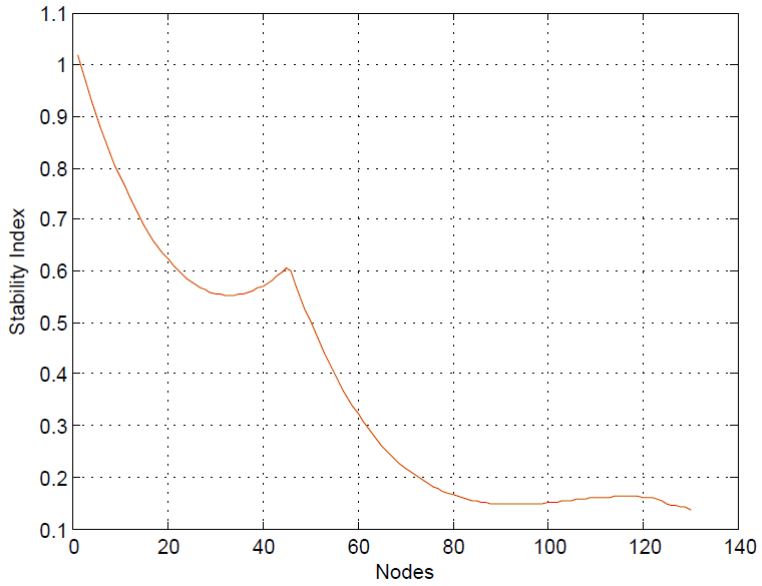


Figure 6.23: Stability Indices for RRT and RRT* without Stability. Green represents RRT and red represents RRT*.

The red path for RRT* lies on top of the green path of RRT as shown in Figure 6.21, 6.22 and 6.23. The stability indices for both planners stayed above zero which is expected. Including a stability check when planning a path did not improve the stability of the system or the path length for RRT*. There might be stable goal nodes that for the arm to get to, it might have to go through some unstable configurations since these planners are random. So adding the stability just ensures that every configuration in the path is stable, it does not increase the stability of the system although a stability check that accomplishes that can be implemented. The path for RRT improved in this case but this has nothing to do with adding a stability check. This is due to the fact that the planner is random and sometimes will randomly produce an optimal path.

In the fourth simulation experiment the same workspace and joint space goals were used as the previous experiment. This time an unstable joint space goal was chosen and the two planners could not find a stable path to the unstable configuration. This shows that the two planners are able to avoid tip-over instability.

The same procedure above was repeated for a workspace goal position of $(x, y, z) = (-0.40, 0.60, 1.00)$ whose chosen stable and unstable joint space goals were $(2.42, -0.44, 0.47, -2.14, 0.8)$ and $(-0.77, -2.33, 0.91, -2.21, 1.72)$ with stability indices of -0.03 and 0.18 respectively. The planners were run for $N = 5000$ iterations and were not terminated when the goal was found. Firstly, the unstable joint space goal was passed to the planners and stability was not considered when planning a path from the home configuration and the results are shown in Figure 6.24, 6.25 and 6.26. Secondly, a stable joint space goal was used in the planning phase without considering stability in the path and the results are shown in Figure 6.27, 6.28 and 6.29. Next, a stable joint space goal was used in the planning phase for planning a path considering the stability of the system in the path and the results are shown in Figure 6.30, 6.31 and 6.32. Lastly, an unstable joint space goal was used in the planning phase and the planners were supposed to plan a stable path to the goal. Again, no stable path was found to the unstable goal configuration as expected.

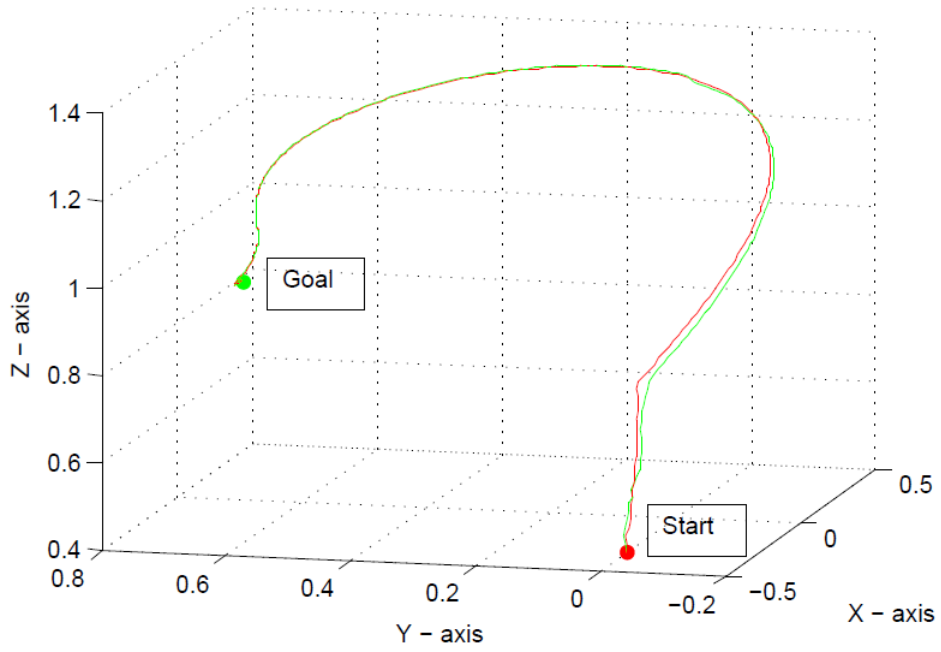


Figure 6.24: End-Effector trajectories for RRT and RRT* without Stability (View 1). Green represents RRT and red represents RRT*.

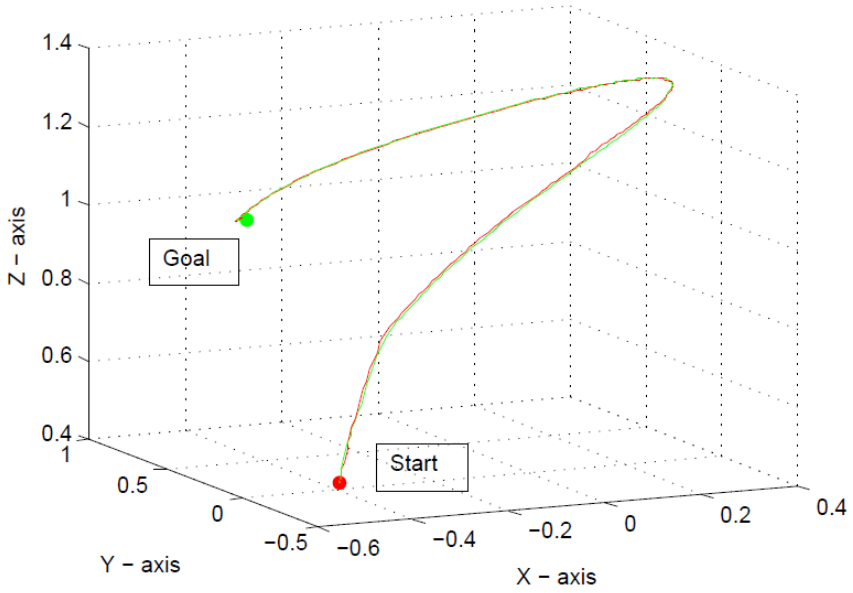


Figure 6.25: End-Effector trajectories for RRT and RRT* without Stability (View 2). Green represents RRT and red represents RRT*.

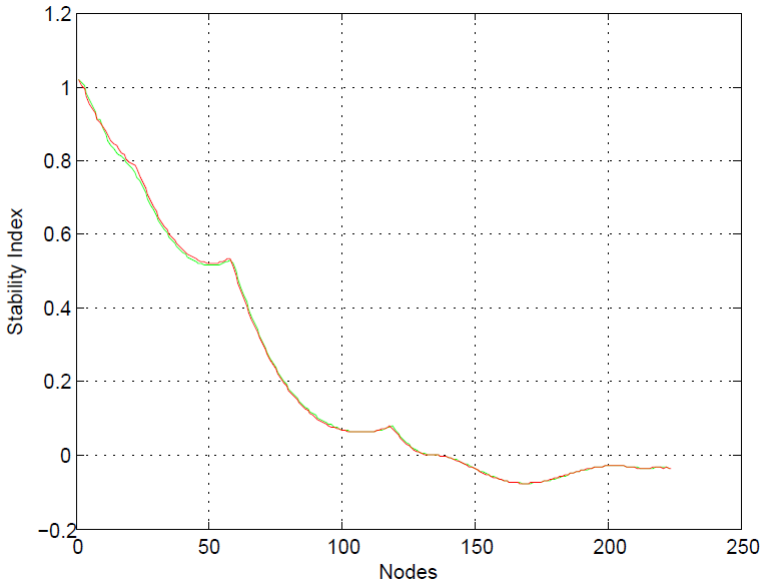


Figure 6.26: Stability Indices for RRT and RRT* without Stability. Green represents RRT and red represents RRT*.

The workspace distance for RRT* in Figure 6.24 and 6.25 was 2.62 meters and for RRT it was 2.63 meters. The joint space distance of RRT* was 4.60 radians and for RRT was 4.62 radians. The workspace distance of RRT* is 0.01 meters shorter than RRT and the joint space distance is 0.02 radians shorter than RRT and again not significantly shorter. RRT* took CPU time of 1 second and RRT took CPU time of 4.77 seconds to plan a path. The system became unstable along the way as the stability indices went below zero at some point as shown in Figure 6.26.

The workspace distance for RRT* in Figure 6.27 and 6.28 was 0.94 meters and for RRT it was 0.97 meters. The joint space distance of RRT* was 2.83 radians for RRT it was 2.84 radians. The workspace distance of RRT* is 0.03 meters (30 mm) shorter than RRT and the joint space distance is 0.01 radians shorter than RRT. In this case RRT randomly chose a longer and a bit jagged path as can be seen in Figure 6.28. RRT* took CPU time of < 0 ms and RRT took CPU time of 230 ms to plan a path. The system showed to be stable along the path for both planners as shown by stability indices of greater than zero in Figure 6.29.

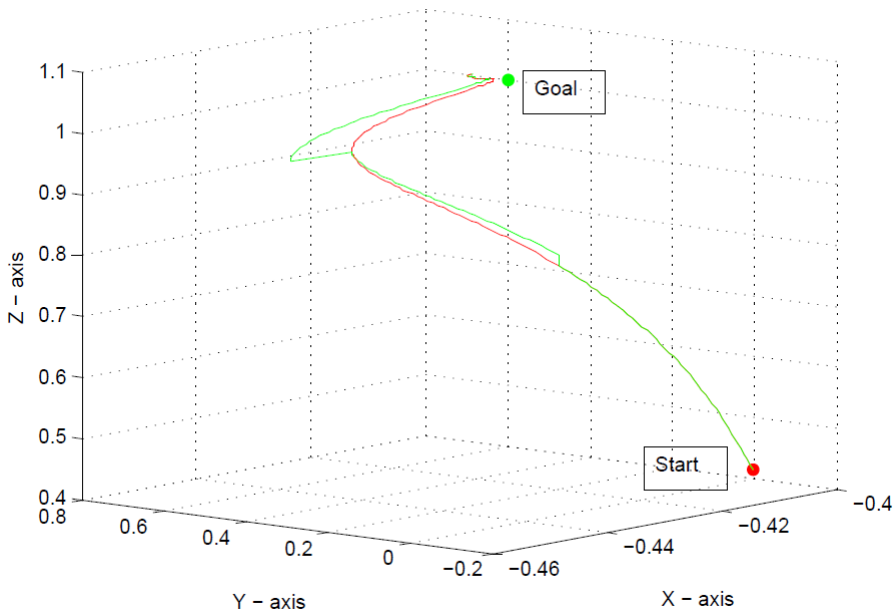


Figure 6.27: End-Effector trajectories for RRT and RRT* without Stability (View 1). Green represents RRT and red represents RRT*.

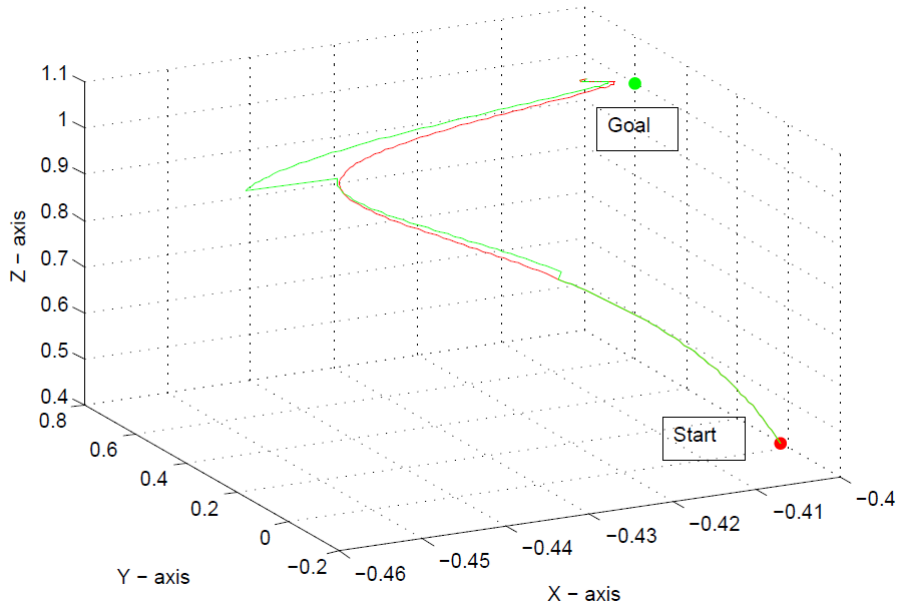


Figure 6.28: End-Effector trajectories for RRT and RRT* without Stability (View 2). Green represents RRT and red represents RRT*.

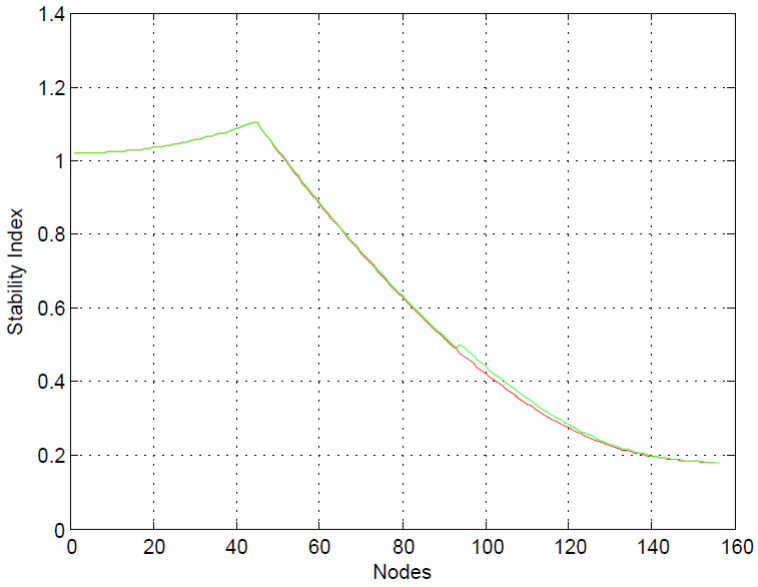


Figure 6.29: Stability Indices for RRT and RRT* without Stability. Green represents RRT and red represents RRT*.

Figure 6.28 shows that the path did not actually reach the desired position (green dot). This happens if the inverse kinematics solution chosen is not accurate, it has been shown in section 6.2 that some inverse kinematics solutions are not accurate, fortunately there is an option to choose other solutions. One solution to this problem would be to filter out all solutions that are not acceptable before making the selection of solutions to use as goal nodes.

In Figure 6.30 and 6.31 both planners produced the same path in workspace and joint space, which are 0.94 meters and 2.83 meters respectively. This is the reason the red path is on top of the green path. RRT* took CPU time of 2 seconds and RRT took CPU time of 1.61 seconds. The same inaccurate inverse kinematics solution was used in this case. The system was stable along the path for both planners. RRT* seems to be slower by a factor of approximately 1.24 when stability checks are introduced.

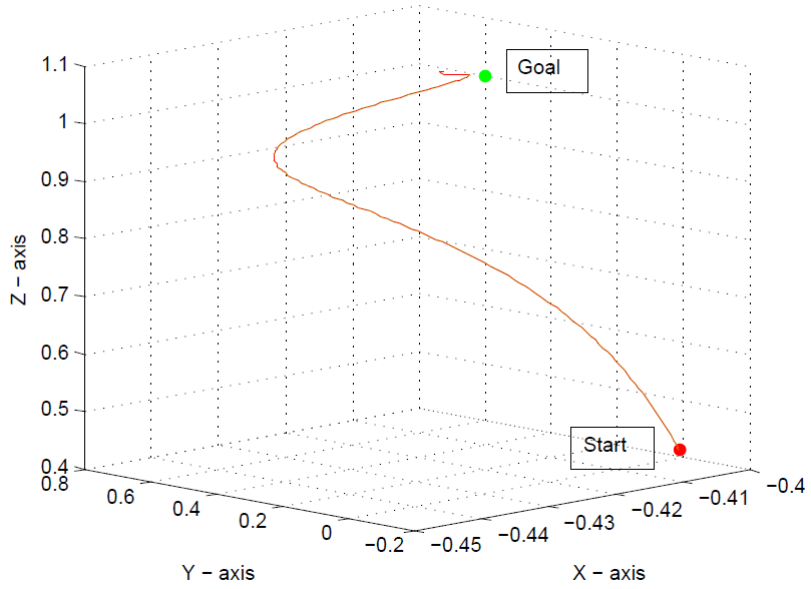


Figure 6.30: End-Effector trajectories for RRT and RRT* without Stability (View 1). Green represents RRT and red represents RRT*.

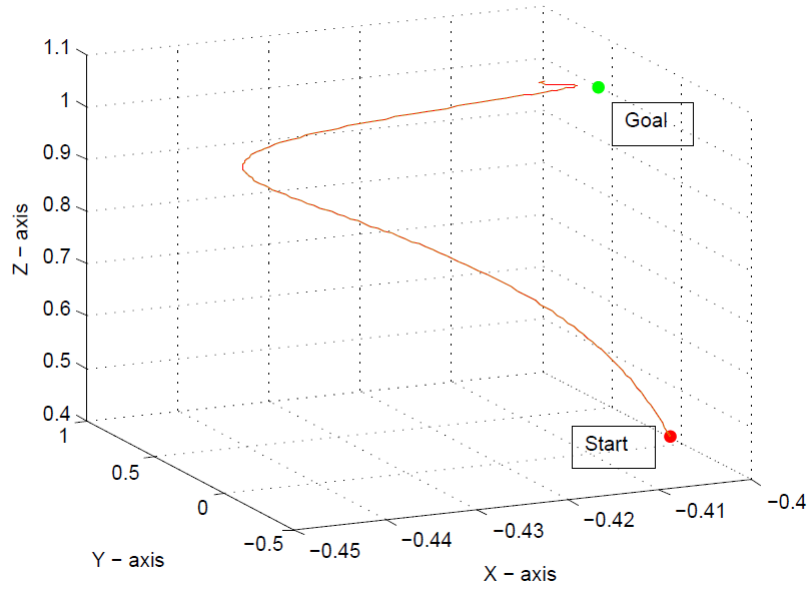


Figure 6.31: End-Effector trajectories for RRT and RRT* without Stability (View 2). Green represents RRT and red represents RRT*.

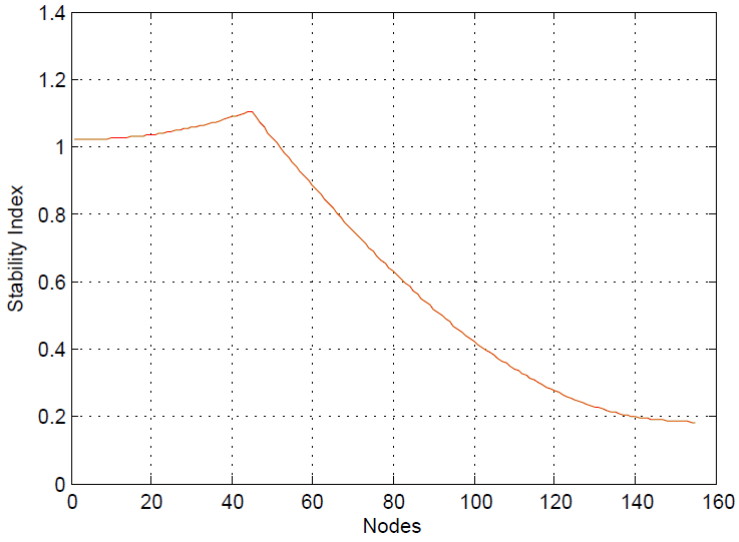


Figure 6.32: Stability Indices for RRT and RRT* without Stability. Green represents RRT and red represents RRT*.

Generally speaking, the RRT* tries to take the shortest path all the time and that is why it produces a shorter path most of the time. It takes longer than RRT to plan the path because it has a rewiring phase that RRT does not have. RRT on the other hand is only concerned with finding a feasible path and it so happens that the path it finds for this particular robot arm is short. In this case the RRT* cannot improve that path any further. However, whether the trajectories produced by these planners are optimal in terms of distance covered or not cannot be easily proven. It seems that whenever a stability check is introduced to the RRT, the path distance decreases and becomes the same as that produced by RRT*. Although the RRT * is slower than RRT it is preferred to be used for safety inspections in this project because it will always strive to produce a shorter path. RRT will sometimes produce a longer and jagged path which is unacceptable when doing safety inspections as the arm can move in an unpredictable manner.

6.6 Safety Inspections Simulation

The goal of this project as mentioned before is to develop and implement algorithms on the Packbot510i manipulator so that it can perform safety inspections in the South African gold mines. This process includes positioning the end-effector of the manipulator in a desired position close to the hanging wall. This task is considered successful if the system carries it out without tipping over. Literature shows that at the time of the project, a robotic system capable of carrying out mining safety inspections had never been developed. In this research area, the focus so far has been on developing systems that can build a map in a mining environment and are capable of localizing themselves within that map. Some research has been done on modeling and control of robot manipulators in simulated and controlled environments such as factories. For the mining safety inspection, research on modeling kinematics of robot manipulators, path planning and tip-over stability measure of mobile manipulators was conducted.

A study was done to evaluate techniques for modeling the kinematics of a 5 DOF redundant manipulator. Closed-form methods were chosen over numeral methods. This is mainly due to the fact that a closed-form technique is able to readily identify all possible solutions of the inverse kinematics problem. Numerical methods are iterative which means they can be slower and in some cases they do not compute all possible or acceptable solutions. Closed-form methods include algebraic modeling and geometric modeling and an algebraic method was not used because the solutions provided do not give a clear indication of how to select the correct solution from the several possible solutions for a particular arm configuration. A geometric solution was chosen because it can readily identify all possible solutions and gives the freedom to choose a solution satisfying system stability and minimum angles movement criteria. Simulation results of the developed geometric model of the Packbot manipulator in Chapter 3 are presented in section 6.2. Results obtained show that this model suffers from computational errors and as a result the computed end-effector position is off-set from the desired end-effector by a maximum of 5.14%. Since these errors are low, this model can be used to mimic the behavior of the real system. Experimental results showing how this model performs on predicting the behavior of the real system are

presented in Chapter 7.

Sampling-based RRT planning algorithms were preferred because their incremental nature allows termination of the algorithm as soon a solution is found as opposed to constructing a tree for the entire environment. This reduces the computational burden of the planning algorithm. Based on the results obtained in Chapter 4, in a 2D case RRT* outperforms RRT Ball and RRT in terms of the cost of the path produced with RRT Ball coming in second place. The RRT* is able to find a shorter path quicker than RRT Ball while RRT produces a non-optimal path. The three planning algorithms were implemented on the model of the Packbot manipulator and compared in section 6.3. Results obtained show that RRT Ball is computationally expensive and is not suitable for use in the mining safety inspections project. In terms of the cost of the path produced by the three planners, RRT* and RRT Ball did not offer any improvement over RRT as was the case in a 2D environment.

Several pieces of research has been carried out for tip-over stability measures, including Zero Moment point, Moment Height Stability and Force Angle stability measures which are widely used. The three stability measures were compared in [4] using real-world data on a mobile robot tipping over to verify that these algorithms accurately match real-world behavior. It was concluded that if noise can be significantly reduced, then the preliminary real-world data suggests that the Force Angle and Moment Height Stability algorithms are able to assess robot stability and can be used as part of a tip-over avoidance system. A Force Angle stability measure model for the Packbot robot was developed in [6] and the source code was readily available for use, therefore, the Force Angle stability measure model was chosen to be implemented in this project. Simulation results obtained in section 6.4 show that the Force Angle stability measure model is able identify configurations that are likely to tip-over the system and adjust their stability indices accordingly.

These three components of the project namely, kinematics, path planning and tip-over stability measure were combined together to form a complete system and were tested in section 6.5. The goal was to position the end-effector in a desired position. Results obtained show that RRT* is slower than RRT by a factor of about 1.24 when the stability measure is introduced. However, this

is compensated by the consistency of the trajectories produced by this planner because it always strives to plan a shorter path. It was found that RRT is not always reliable and can sometimes produce a longer and jagged path which would result in a robot behaving in an unpredictable manner during operation. For this reason, RRT* was chosen as a planning algorithm to be used in the safety inspections simulation. It was observed that the accuracy of a planning algorithm depends on the accuracy of the inverse kinematics solutions provided to it. The end-effector trajectories presented in several figures such as 6.22 show that the path produced by the planners is smooth and does not need smoothing in this case.

In this section, all three components, kinematics model, Force Angle stability measure and RRT*, are integrated together to form a complete robotic system that is capable of performing safety inspections in the mine. The input to this system is the (x, y, z) in the Packbot frame and the output is the path from the start position to the end position in joint space. The goal position in Cartesian frame is sent to the inverse kinematics node, which transforms this into a number of arm configurations in joint space. The Cartesian frame of the system is shown in Figure 6.34 with the Z-axis pointing out of the page. All configurations with stability indices less than zero are discarded and out of the stable configurations, the configuration with the minimum joint change from the current configuration is chosen as the goal. The joints that are considered when choosing a goal configuration are $\theta_1, \theta_2, \theta_3$ and θ_4 the reason being, if these joints are allowed to make big steps the path becomes too long and jerky and the risk of tipping over becomes high. θ_5 does not have much effect on the stability of the system and the length of the path taken. Once the goal configuration has been chosen it is sent to the planning node. The planning node computes a stable path from the start configuration to the goal configuration. It does this by making sure that all configurations in the path have stability indices greater than zero. If it cannot find a set of configurations with stability indices greater than zero then it returns a no path found message.

The above mentioned procedure in summarized is the steps below:

- Retrieve goal (x, y, z) position.

- Compute Inverse Kinematics solutions.
- If no solution, report no IK solution, otherwise continue.
- Compute the stability indices of all solutions.
- Check the current configuration of the arm.
- Select a stable configuration that has minimum change from the current configuration as a goal.
- Send the goal to the planning node.
- Check the current configuration of the arm.
- Plan a stable path from the current configuration to the given goal configuration.
- If no stable path found, report no path, otherwise return the path.

The procedure explained above was followed to carry out a safety inspection simulation while the mobile base was stationery. The robot arm was only allowed to operate under a 1 meter roof. Figure 6.34 shows a region in the XY frame that the robot arm was able to reach out without tipping over the system and with all the links operating under a 1 meter roof. The end-effector desired height was kept at 0.98 meters above the ground. A different end-effector desired height would result in a slightly different operational region. The height of 0.98 was chosen as to compensate for the inverse kinematics errors that may arise during the inspection. This operation was done manually testing all the points that the arm can move to in a fashionable way without violating the constraints of the environment and the robot system. In Figure 6.34, the red rectangular box represents the base of the robot and the blue line is the boundary of the arm. Looking at the negative side of the x - axis, the furthest the arm can reach is -0.68 meters and on the positive side it is 0.66 meters. In the positive y - axis the furthest it can go is 0.89 and on the negative y - axis it is -0.88 meters.

The selection of the inverse kinematics solution based on the system stability and the minimum change in joint angles play a big role in the shape of the

operational region. The height constraint is also another factor. The region on the left side of the base frame is almost symmetrical about the y - axis. The base frame is where the arm starts and is shown in magenta in Figure 6.34 and the center of the robot, known as the packbot frame is shown in blue. The reason for that is all the stable solutions in that region are similar in the formation of the arm as shown in Figure 6.35. The joint angle that varies a lot in these configurations is θ_1 as the arm moves sideways and the rest vary as the arm moves away from the base frame. These configurations are closest to the home configuration in terms of total angle difference. The home configuration is shown in Figure 6.33. Now on the right side of the base frame, the region is not close to being symmetrical about the y - axis. The reason for this is that the solutions selected are not similar in formation. The formation of the region above the y - axis is shown in Figure 6.36. This formation has a smaller operating region because as the end-effector goal moves further away from the base frame, the end-effector tends to go above the roof. The region between $y = 0$ and $y = -0.37$ is almost symmetrical about the y - axis because the arm changes from the configuration in Figure 6.36 to the configuration in Figure 6.35 and now its formation is also the same as that on the $y > 0$ region. For the region below $y = -0.37$, the formation of the solutions chosen is also similar to one shown in Figure 6.35.

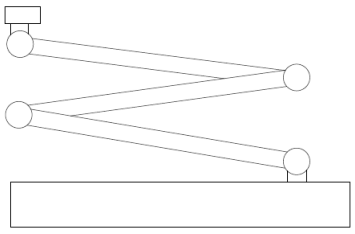


Figure 6.33: Packbot Manipulator home configuration.

This section shows how this system was used to position the end-effector in a desired position in a simulated environment. The system was limited to a height of 1 meter and any part of the manipulator was not supposed to hit the ground. Simulation results presented in this section show that the developed robotic system is capable of performing safety inspections in a simulated environment without tipping over the system.

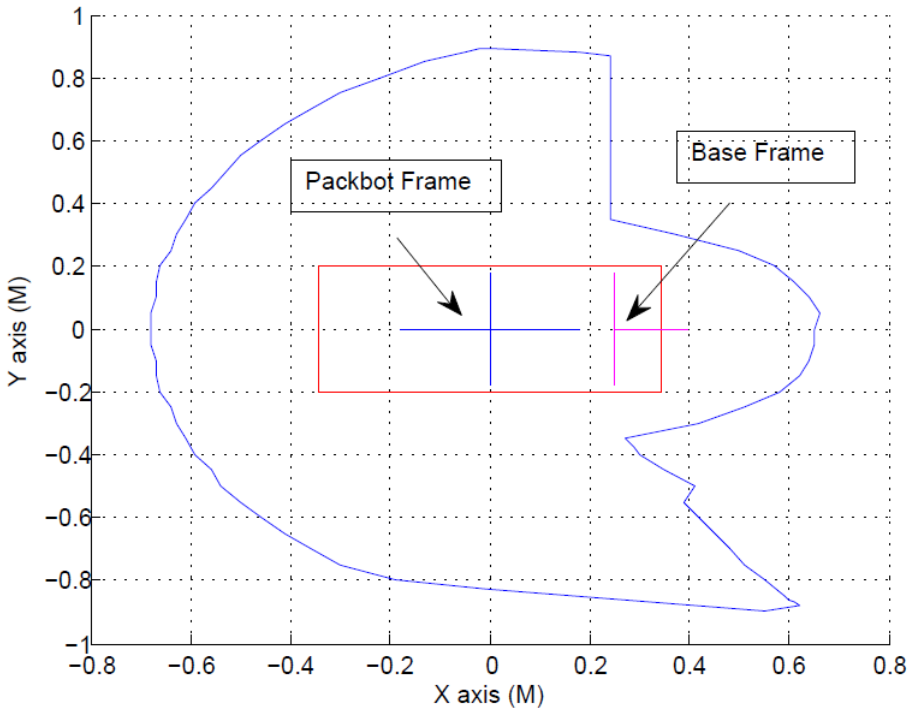


Figure 6.34: Operational region for the safety inspections simulation under a height restriction of 1 meter and end-effector goal height of 0.98 meters.

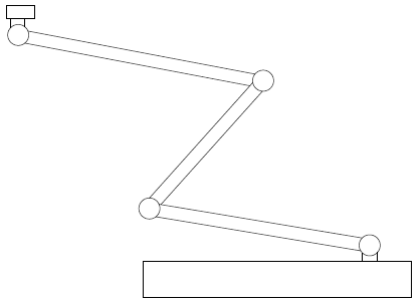


Figure 6.35: Packbot Manipulator formation 1. This formation places the end-effector behind the base of the robot.

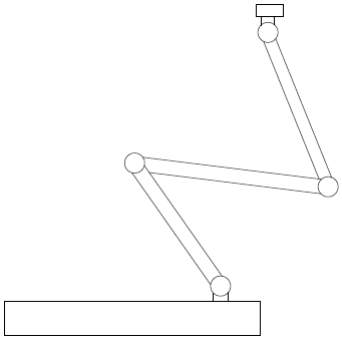


Figure 6.36: Packbot Manipulator formation 2. This formation places the end-effector in front of the base of the robot.

Figure 6.34 shows a region that the manipulator is able to cover while standing in one position. However, it has been observed that although this manipulator can successfully complete the task, using a shorter arm with less degrees of freedom would give a bigger operational region. This robot arm is too long for this task and in most cases it has to unfold itself before it can reach the desired end-point. During this unfolding process, the arm would either hit the ceiling or tip-over the system thus reducing the operational region.

Chapter 7

Experimental Results

Experimental results of the tests performed on the selected methods are presented here. Tests were done on a Dell XFR running Linux Ubuntu 12.04(precise) with 3.8 GB RAM and an Intel(R) Core(TM) i7-2640M CPU 2.8 GHz. The fuerte version of ROS was installed on this system. Experimental results on the kinematics modeling are presented in section 7.1 together with their analysis. Section 7.2 discusses the implementation of RRT* on the real system, i.e., the iRobot Packbot510i and evaluates its limitations. Only RRT* is implemented on the real system because it was chosen as a suitable planning node for the safety inspection simulation in Chapter 6. The tip-over stability measure experimental results are presented in section 7.3 with their discussions and analysis. This will show how the Force Angle stability measure performs in terms of predicting the actual tip-over instability.

Experimental results of the complete system are then presented combining kinematics, planning and tip-over prevention. Finally, experimental results showcasing the performance of the mine safety inspections system are provided, together with a discussion of overall system limitations.

7.1 Kinematics

In this section, the forward kinematics and inverse kinematics developed in Chapter 3 are tested on the real system to validate their performance. To evaluate how

the forward kinematic model matches the real system, the robot arm was put in different configurations and the joint positions were recorded together with their corresponding end-effector positions. The measured joint positions were used to calculate the end-effector positions using the forward kinematics developed in Chapter 3 and the results are compared with the measured end-effector positions to see how close the model is to the real system. Table 7.1 contains the measured joint angles together with their corresponding end-effector positions. The measurements were taken using the robot internal sensors and the iRobot software interface. Table 7.2 contains the calculated end-effector positions using the forward kinematics model with the errors on each axis and the total error, i.e, $Error_{tot} = \sqrt{(X_r - X_m)^2 + (Y_r - Y_m)^2 + (Z_r - Z_m)^2}$ where r represents the real system and m is the model.

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	0.06	-2.60	2.43	-2.84	1.49	-0.20	-0.09	0.72
2	-0.74	-2.66	2.22	-2.57	1.44	-0.19	0.39	0.86
3	1.02	-2.77	1.86	-1.66	1.06	-0.02	-0.59	1.29
4	0.28	-2.01	1.74	-1.68	0.45	0.44	-0.02	1.49
5	-0.69	-1.79	1.50	-1.43	0.29	0.55	-0.33	1.58
6	1.26	-2.09	2.16	-2.08	0.51	0.36	0.09	1.24
7	1.26	-1.53	2.16	-2.08	0.05	0.52	0.59	1.03
8	0.39	-1.25	1.56	-1.49	-0.48	1.23	0.33	1.14
9	0.01	-1.16	0.77	-0.75	-0.50	1.34	-0.06	1.56
10	-0.70	-1.07	0.68	-1.12	-0.04	0.91	-0.65	1.59

Table 7.1: Measured joint angles with their corresponding measured end-effector positions

The maximum absolute error on each axis from Table 7.2 is $X_{error} = 0.07$ meters, $Y_{error} = 0.06$ meters and $Z_{error} = 0.04$ meters. Most of the errors on the Z - axis are caused by over-shooting which can cause a problem when there are height restrictions. This means when a height restriction is used the end-effector goal position used in kinematics should be set to be lower than the allowed height as done in the safety inspection simulation in Chapter 6. The maximum total error is 0.07 meters, this suggests that the model is off-set from the real system by a maximum end-effector distance of 7.00 centimeters. This is expected to be

Num	X	Y	Z	X_{error}	Y_{error}	Z_{error}	$Error_{tot}$
1	-0.24	-0.04	0.74	0.04	-0.05	-0.02	0.06
2	-0.18	0.37	0.88	-0.01	-0.06	0.02	0.06
3	-0.09	-0.59	1.31	0.07	0.00	-0.02	0.07
4	0.40	0.03	1.53	0.04	-0.04	-0.04	0.07
5	0.56	-0.27	1.62	-0.01	-0.05	-0.04	0.07
6	0.29	0.09	1.28	0.06	0.01	-0.04	0.07
7	0.46	0.59	1.08	0.06	-0.01	-0.04	0.07
8	1.21	0.38	1.19	0.02	-0.05	-0.04	0.07
9	1.34	-0.01	1.59	0.00	-0.05	-0.04	0.07
10	0.95	-0.62	1.63	-0.04	-0.03	-0.04	0.06

Table 7.2: Computed end-effector positions with their corresponding errors.

the case since the actual dimensions of the robot were not provided and they had to be measured and estimated. Another thing to note is that modeling a real system is a difficult task to do and a model cannot match a real system exactly.

To validate the inverse kinematics model, some of the data used in section 6.2 will be used here. In the simulation of the inverse kinematics model, certain end-effector goal positions were chosen and their inverse kinematics solutions were computed. A sample of these solutions was selected and their corresponding end-effector positions were computed and compared with the desired end-effector position. In this section, some of these solutions are chosen and are executed on the real system to see if the robot end-effector position will be the same as the desired end-effector position.

Table 7.3 shows some of the inverse kinematics solutions taken from section 6.2 and their corresponding desired end-effector positions. These solutions were executed on the real system and joint positions on the real system were recorded together with their corresponding end-effector positions and are shown in Table 7.4.

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	1.26	-1.19	1.89	-2.23	0.17	0.50	0.70	1.00
2	4.34	-1.79	-2.10	2.08	0.17	0.50	0.70	1.00
3	-0.90	-1.51	-1.10	-0.19	-1.10	-0.50	0.90	1.20
4	-0.90	-1.90	-0.22	-1.19	1.43	-0.50	0.90	1.20
5	2.29	-1.63	1.10	0.19	1.10	-0.50	0.90	1.20
6	-0.43	-1.56	-1.38	-0.17	-1.45	-0.90	0.50	0.80
7	-0.43	-2.00	-0.46	-1.31	1.9	-0.90	0.50	0.80
8	0.74	-1.81	-0.91	0.81	1.68	-0.70	-0.90	0.80
9	3.92	-1.49	1.29	0.16	1.45	-0.70	0.90	0.80
10	0.69	-0.56	0.84	-2.40	0.81	0.90	0.50	0.90
11	3.76	-2.35	-1.40	2.40	-0.44	0.90	0.50	0.90

Table 7.3: Inverse Kinematics Solutions executed on the real system with their corresponding desired end-effector positions in radians.

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	1.26	-1.24	1.89	-2.23	0.21	0.54	0.67	0.99
2	-1.94	-1.79	-2.06	2.08	0.21	0.46	0.70	0.98
3	-0.89	-1.51	-1.07	-0.20	-1.08	-0.51	0.86	1.24
4	-0.90	-1.85	-0.26	-1.18	1.39	-0.49	0.84	1.24
5	2.29	-1.62	1.06	0.19	1.15	-0.46	0.92	1.22
6	-0.43	-1.59	-1.31	-0.17	-1.46	-0.93	0.47	0.84
7	-0.43	-1.99	-0.48	-1.31	1.86	-0.93	0.47	0.79
8	0.73	-1.82	-0.88	0.81	1.69	-0.68	-0.93	0.85
9	-2.36	-1.49	1.28	0.16	1.43	-0.73	0.87	0.79
10	0.68	-0.61	0.85	-2.39	0.86	0.90	0.45	0.92
11	-2.52	-2.34	-1.39	2.40	-0.45	0.86	0.51	0.88

Table 7.4: Joint Readings from the real system with their corresponding end-effector positions in radians.

Looking at Table 7.3 and 7.4, measured joint values in Table 7.4 are not always the same as the as the joint values executed on the robot shown in Table 7.3. For example, row four in Table 7.3 and 7.4, the absolute differences between the executed joint angles and the measured joint angles in degrees are $(0.0, 2.86, 2.29, 0.57, 2.29)$ and the sum of the angle differences is 8.01 degrees. The resulting absolute end-effector position errors in all axes are $(0.01, 0.04, 0.04)$ and the total end-effector error is 0.06 meters. This indicates that the end-effector of the real system is off-set from the end-effector of the model by about 6 centimeters. This could be a result of the inaccuracy of the encoders used to read joint positions of the robot. This can also arise from the poor performance of the control system used on each joint. Gravity is not compensated for since the dynamics of the arm are not considered in this thesis, this also has an effect on the errors observed. Measured value of θ_1 in row two, nine and eleven in Table 7.4 is not the same the executed value. Geometrically the arm ends up in the same position but takes different directions to get there. For example, row eleven of Table 7.4, θ_1 is -2.52 radians which is equivalent to -144.39 degrees. Row eleven of Table 7.3, θ_1 is 3.76 which is equivalent to 215.43 degrees. The angle 215.43 degrees and angle -144.39 degrees lie in the same quadrant and in fact are the same angles measured in different directions. This is due to the existing planner in the iRobot software interface. It normally executes the joint angle that results in the least movement of the joint, 215.43 degrees moves the joint more than -144.39 degrees, hence -144.39 degrees was executed in this case. This causes problems when θ_1 joint makes big changes but since a configuration that results in less movements of the arm is chosen in the safety inspections process, this will not affect the results obtained.

Experimental results obtained in this section indicate that the model does not match the real system exactly as expected. Measured joint angle values are not always the same as the executed joint angle values. The dynamics of the system are not taken into account which has an effect on the errors observed. However, this model can be used to develop planning algorithms for the manipulator since the errors observed are as small as 7 centimeters.

7.2 Planning

Experimental results showcasing the performance of the RRT* are discussed in this section. This is done by first driving the arm to an arbitrary position and record the joint positions and the end-effector position. The arm is then driven to its home configuration and the planner is used to plan a path from this home configuration to the recorded configuration. This is done to study how the robot arm responds to the planner. Table 7.5 shows the arbitrary joint positions together with their corresponding end-effector positions and Table 7.6 shows the corresponding joint positions after the path has been executed on the robot together with their corresponding end-effector positions. It is important to mention that the iRobot software has its own control system that controls the arm. This control system does not always respond to the input of the system as observed from Table 7.3 and 7.4 in section 7.1.

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	-0.84	-2.77	2.16	-2.22	0.45	-0.23	0.44	0.99
2	0.74	-2.72	2.14	-2.13	1.24	-0.09	-0.41	1.08
3	0.74	-1.63	1.74	-2.71	1.66	0.39	0.04	1.04
4	-0.75	-1.14	1.10	-2.6	1.66	0.49	-0.32	1.06
5	-0.85	-2.85	1.93	-1.23	1.67	-0.07	0.26	1.39
6	-0.85	-2.83	1.12	-0.63	1.06	-0.48	0.73	1.49
7	0.13	-2.36	-0.01	0.01	1.02	-1.00	-0.23	1.52
8	1.08	-2.72	2.11	-1.84	0.99	0.09	-0.43	1.23
9	2.45	-2.94	1.43	-0.99	0.99	1.07	-0.86	1.36
10	0.08	-1.41	1.09	-1.99	0.98	0.57	-0.04	1.45

Table 7.5: Arbitrary joint positions with their corresponding end-effector positions. Measurements taken from the real system.

Comparing Table 7.5 and Table 7.6, it can be seen that joint angles executed on the system, with their corresponding end-effector position, from the output of the planner are not always the same as the measured joint angles used as input to the planner. For example, row one of Table 7.5 and 7.6, the absolute difference in degrees between these two configurations is $(2.29, 2.29, 1.72, 4.01, 0.00)$ and the sum of the total angle difference is 10.31 degrees. Figure 7.1 to 7.6 show the plot

Num	θ_1	θ_2	θ_3	θ_4	θ_5	X	Y	Z
1	-0.80	-2.73	2.19	-2.29	0.45	-0.22	0.39	0.96
2	0.71	-2.69	2.18	-2.22	1.21	-0.09	-0.39	1.05
3	0.72	-1.70	1.82	-2.75	1.66	0.35	-0.00	1.01
4	-0.74	-1.21	1.24	-2.65	1.66	0.47	-0.29	1.04
5	-0.80	-2.76	1.97	-1.40	1.67	-0.04	0.21	1.37
6	-0.83	-2.82	1.19	-0.74	1.01	-0.47	0.69	1.49
7	0.13	-2.36	0.01	-0.02	0.99	-0.99	-0.23	1.54
8	1.04	-2.67	2.15	-1.96	0.99	0.09	-0.40	1.19
9	2.25	-2.92	1.56	-1.18	0.99	0.88	-0.68	1.34
10	0.08	-1.43	1.13	-2.00	0.99	0.56	-0.04	1.45

Table 7.6: Joint readings after the path, from the real system with their corresponding end-effector positions. Measurements taken from the real system.

of the simulated end-effector position with the measured end-effector position of the robot.

In all figures, the simulated end-effector path is always off-set from the end-effector path of the real system. This is expected and has been explained in section 7.1. In an ideal system, the path of the end-effector from a real system should match that of the simulated system. This means the blue path on the figures should have the same shape as the red path. In Figure 7.1 to 7.3, the path of the real system closely matches the simulated path with small variations due to the vibration of the real system. In all cases the end-effector position of the real system is not exactly the same as the desired position but close. This is because the joint angles of the arm are not always set to the values produced by the planners and this affect the position of the end-effector. In Figure 7.4 and 7.5, the path of the real system although it follows the simulated path, it is more jagged and not smooth. This is due to the inconsistency of the control system of the arm. The trajectory in Figure 7.3 has the same difficulty as the trajectory in Figure 7.4 but the end-effector of the real system in Figure 7.3 is smoother and matches the simulated trajectory better than that in Figure 7.4.

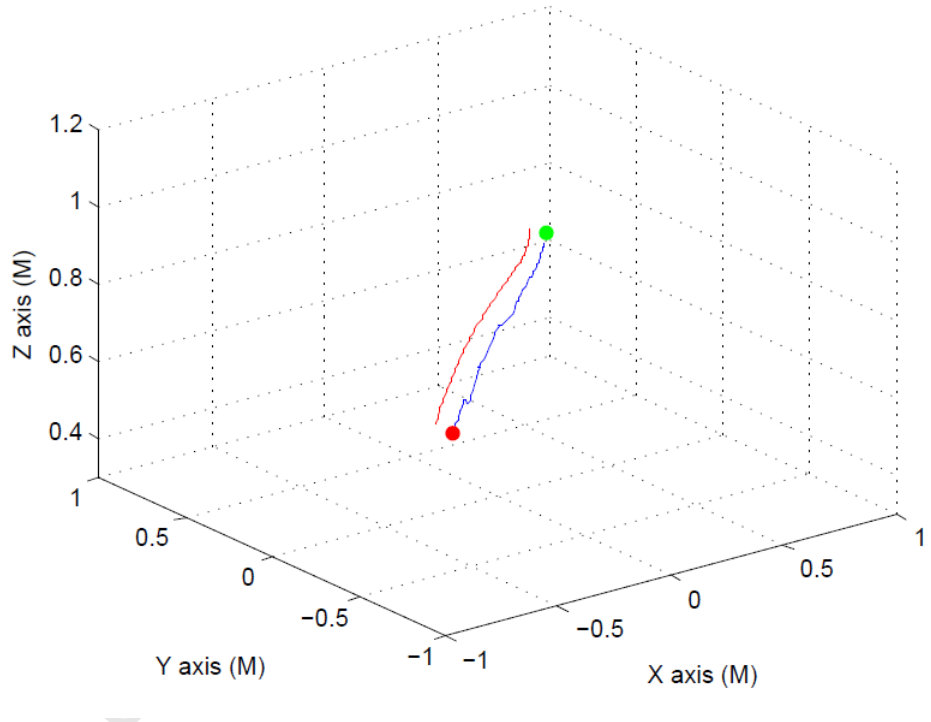


Figure 7.1: Simulated end-effector path vs end-effector path of the real system. Red represents the simulated path and blue is the path from the real system.

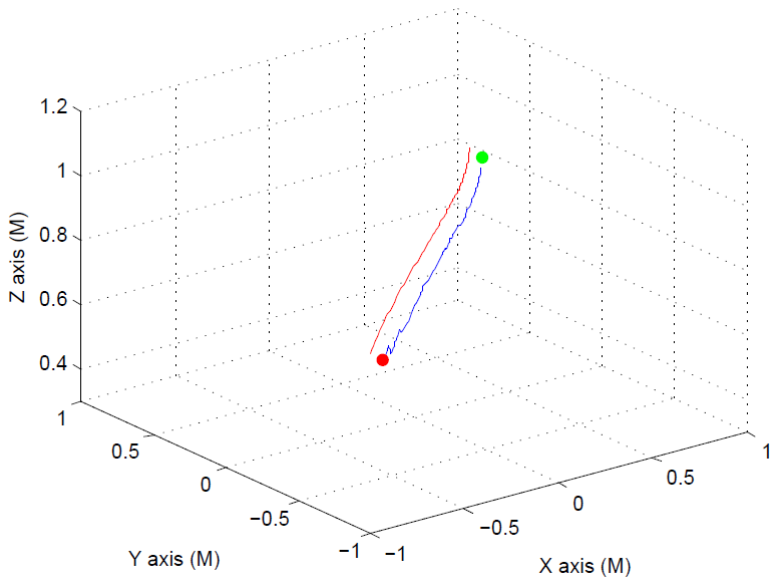


Figure 7.2: Simulated end-effector path vs end-effector path of the real system. Red represents the simulated path and blue is the path from the real system.

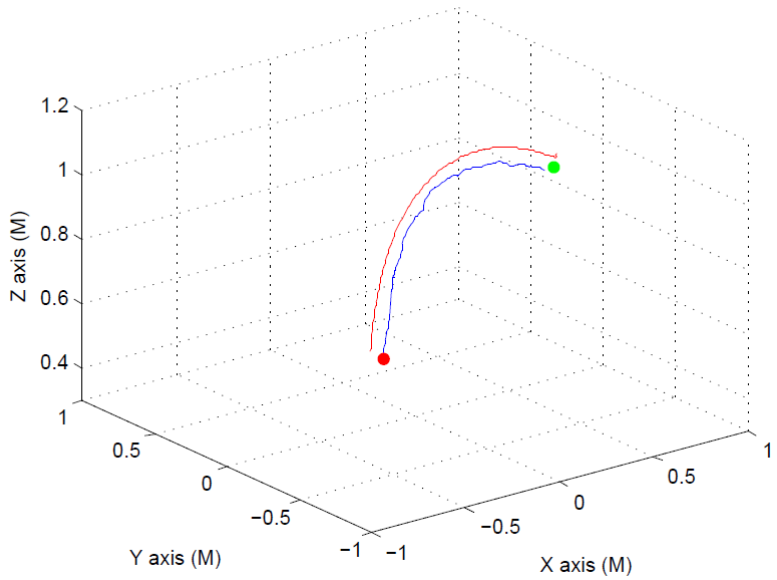


Figure 7.3: Simulated end-effector path vs end-effector path of the real system. Red represents the simulated path and blue is the path from the real system.

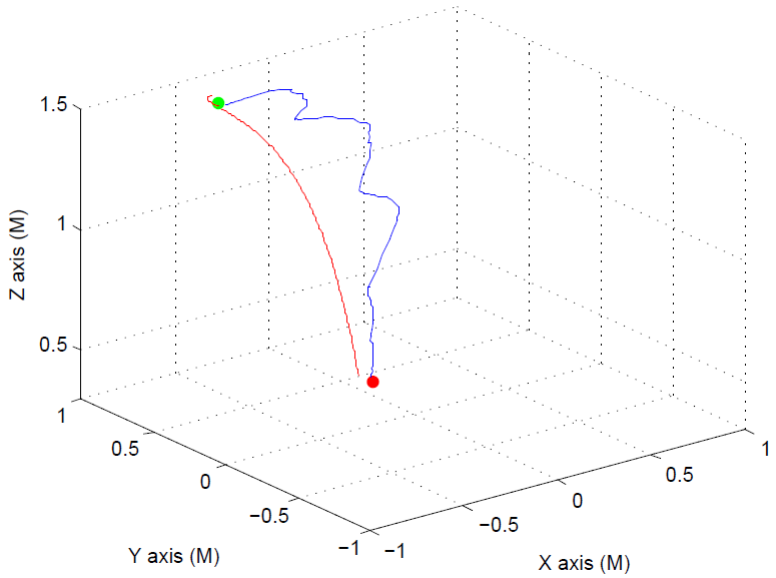


Figure 7.4: Simulated end-effector path vs end-effector path of the real system. Red represents the simulated path and blue is the path from the real system.

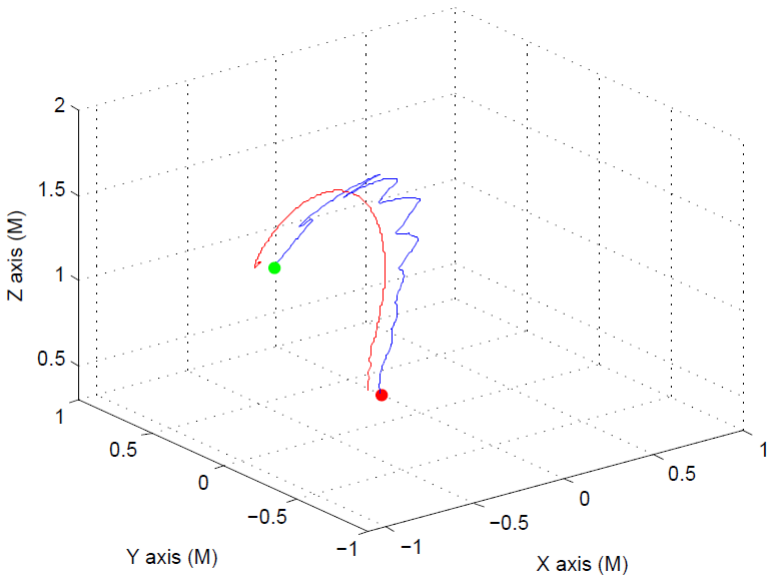


Figure 7.5: Simulated end-effector path vs end-effector path of the real system. Red represents the simulated path and blue is the path from the real system.

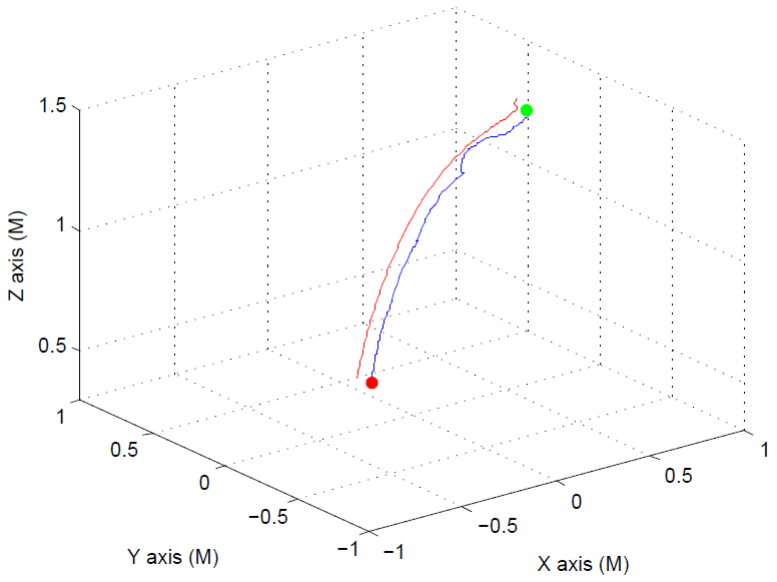


Figure 7.6: Simulated end-effector path vs end-effector path of the real system. Red represents the simulated path and blue is the path from the real system.

In Figure 7.6, the path of the end-effector of the real system follows the simulated path with a small ditch towards the end but is better than that of Figure 7.4 and 7.5. This shows that the control system can sometimes fail to follow a simple path that it previously followed and is inconsistent.

Figure 7.7 shows trajectories of the simulated path and of the real system ran 10 times for the same end-effector goal position. This figure further highlights the inconsistency of the control system of the iRobot software. For the same end-effector goal position, the variation in the simulated path in red is very small compared to the path of the real system. The variation of the path of the real system in blue is very big which proves that the controller is inconsistent.

Experimental results presented on this section show the response of the robot arm to the input given from the RRT* planner. Data in Table 7.5 and 7.6 show that the control system does not always respond to the input given to the system. The path produced by the end-effector of the real system can sometimes follow the simulated path with small vibrations due to vibrations of the system as it moves. In some cases the path of the end-effector of the real system is jagged and

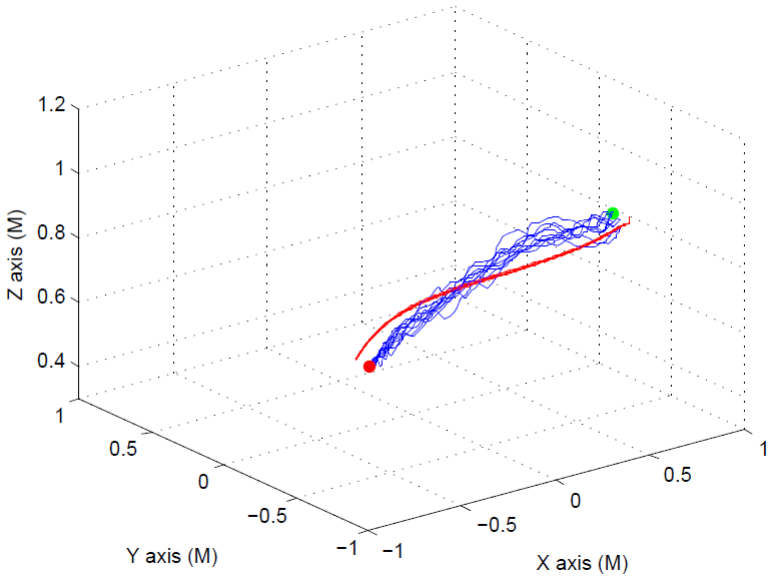


Figure 7.7: Simulated end-effector path vs end-effector path of the real system of the same experiment repeated 10 times. Red represents the simulated path and blue is the path from the real system.

this is because the control system of the robot is unable to respond to the inputs given to it. It was observed that the accuracy of the Packbot manipulator is poor because it does not drive the end-effector to the exact given point in space. It was also observed that the repeatability of the manipulator is poor. Repeatability is a measure of how close a manipulator can return to a previously taught point.

7.3 Tip-over Stability Measure

Experimental results of the Force Angle stability measure developed in Chapter 5 are presented in this section. The stability measure was tested on the real system and the stability index of various configurations was recorded. Figure 7.8 to 7.12 show configurations of the robot arm with their corresponding stability indices.

The stability index of the configuration in Figure 7.8 is -1.56 which indicates that the system is already tipping over or has tipped over. Figure 7.8 shows the robot tipping over towards the front and this is in agreement with the stability index of -1.56.

Figure 7.8: Configuration with joint positions (3.06, -2.71, 0.94, -0.72,1.97) and stability index of -1.56.

Figure 7.9: Configuration with joint positions (-1.72, -2.73, 1.67, -1.49,1.97) and stability index of 0.00.

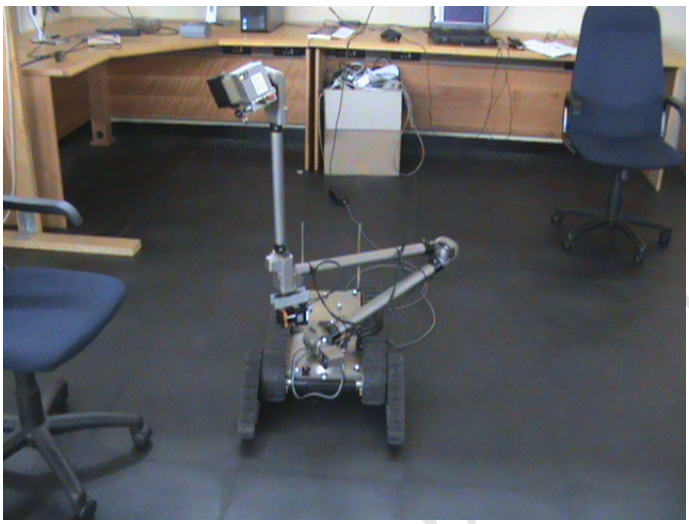
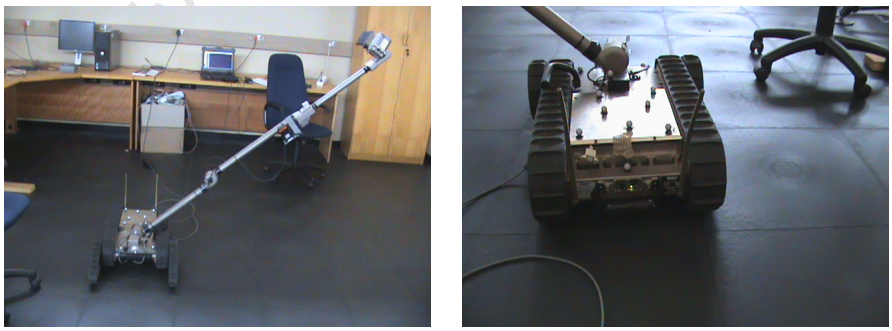


Figure 7.10: Configuration with joint positions (-1.03, -2.76, 2.69, -1.57,1.98) and stability index of 0.85.



(a) Front View (b) Rear View

Figure 7.11: Configuration with joint positions (-1.62, -2.39, -0.01, -0.00,2.02) and stability index of -0.39.

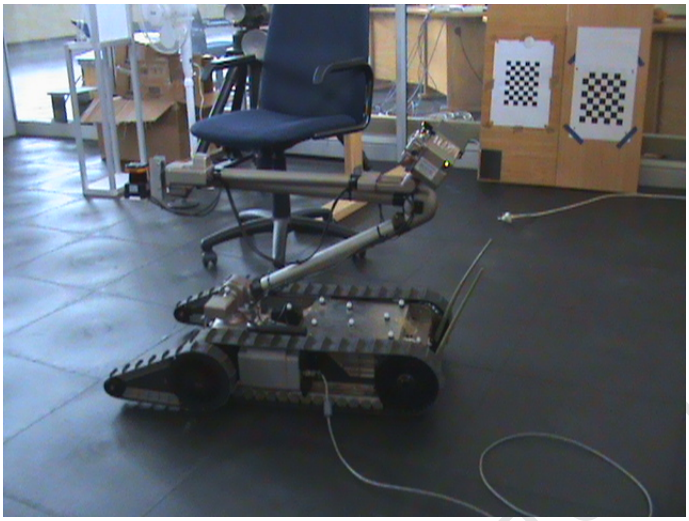


Figure 7.12: Configuration with joint positions (0.00, -2.65, 2.63, -3.12,1.08) and stability index of 0.93.

The stability index of the configuration in Figure 7.9 is 0.00 and this indicates that the system is at a point of tip-over. The Figure in 7.9 shows a robot which is stable but has potential risk of tipping over since the end-effector is away from the center of the base and high above the ground. This moves the center of the mass away from the support polygon. The stability index of the configuration in Figure 7.10 is 0.85 and the end-effector is close to the center of the robot and the majority of the arm is directly above the base which brings the center of mass inside the support polygon. The stability index agrees with the stability of the real system in this case. Figure 7.3 shows the front view and the rear view of the robot tipping over with a stability index -0.39. The center of mass in this case is clearly outside of the support polygon and therefore the stability index is in agreement with the instability of the system. Lastly, Figure 7.12 shows the robot with a configuration close to the home configuration with a stability index of 0.93. As expected the stability index of this configuration is approaching one.

Experimental results of the tip-over stability measure presented in this section indicate that the Force Angle stability measure is able to detect the point of tipping over. Manipulator configurations that are likely to tip-over the system are assigned a stability of index of less than zero and configurations that keep the system stable are assigned a stability index of zero and above. It was observed

that this stability measure is too cautious, i.e, some configurations that have a negative stability index still keep the system stable but are close to tipping over the system. This is good because being cautious means when the system tips over, the tip over point would have been detected earlier. However, this can affect the performance of the system in terms of the operational region during the safety inspections. The effect of the flippers attached to the base on the stability of the system is beyond the scope of this thesis. This experiment does not verify the accuracy of the Force Angle stability measure but the accuracy is of little consequence for the purposes of tip-over prevention.

7.4 Safety Inspections Experiment

Previous sections presented experimental results of the three components namely, kinematics modeling, path planning and tip-over stability measure. Each of these components have their own downfalls with the kinematics model not representing the real system exactly, control system not responding to the input given from the output of the planner and the tip-over stability measure being too cautious. In this section, all three components are combined to form a complete system and using this system the safety inspections experiment is carried out. The goal is to find out how this system performs in carrying out the safety inspections considering the limits of each component.

The procedure followed and the experimental setup are the same as the safety inspections simulations. The robot arm was only allowed to operate under a height restriction of 1 meter and any part of the arm was not allowed to touch or hit the ground. The end-effector height was chosen to be 0.98 meters to compensate for the off-set between the model and the real system. This experiment was carried out manually testing all the points that the manipulator can move to in a fashionable way without violating the constraints of the environment and the robot system. The results obtained from this experiment are compared to the results obtained from the safety inspections simulation in section 6.6.

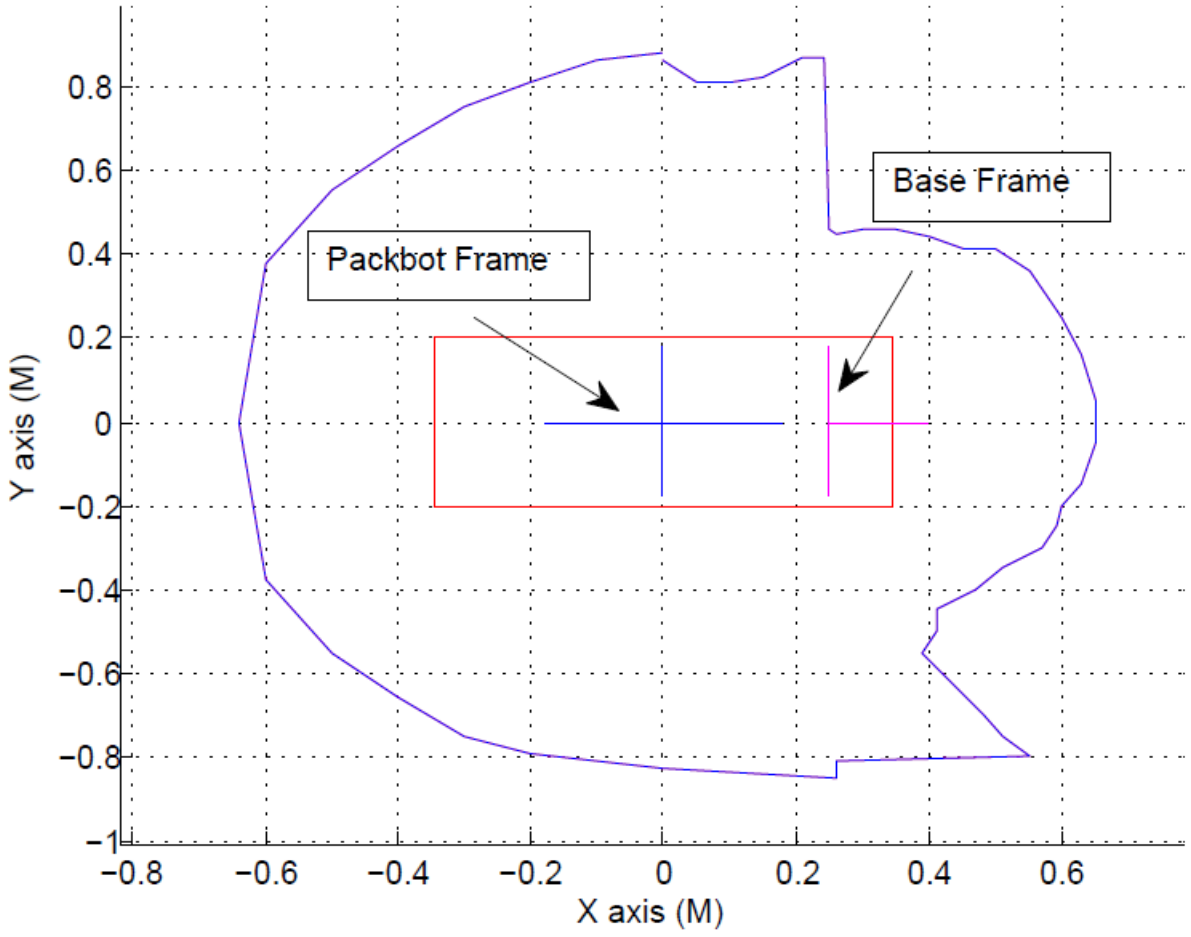


Figure 7.13: Operational region for the safety inspections experiment under a height restriction of 1 meter and end-effector goal height of 0.98 meters.

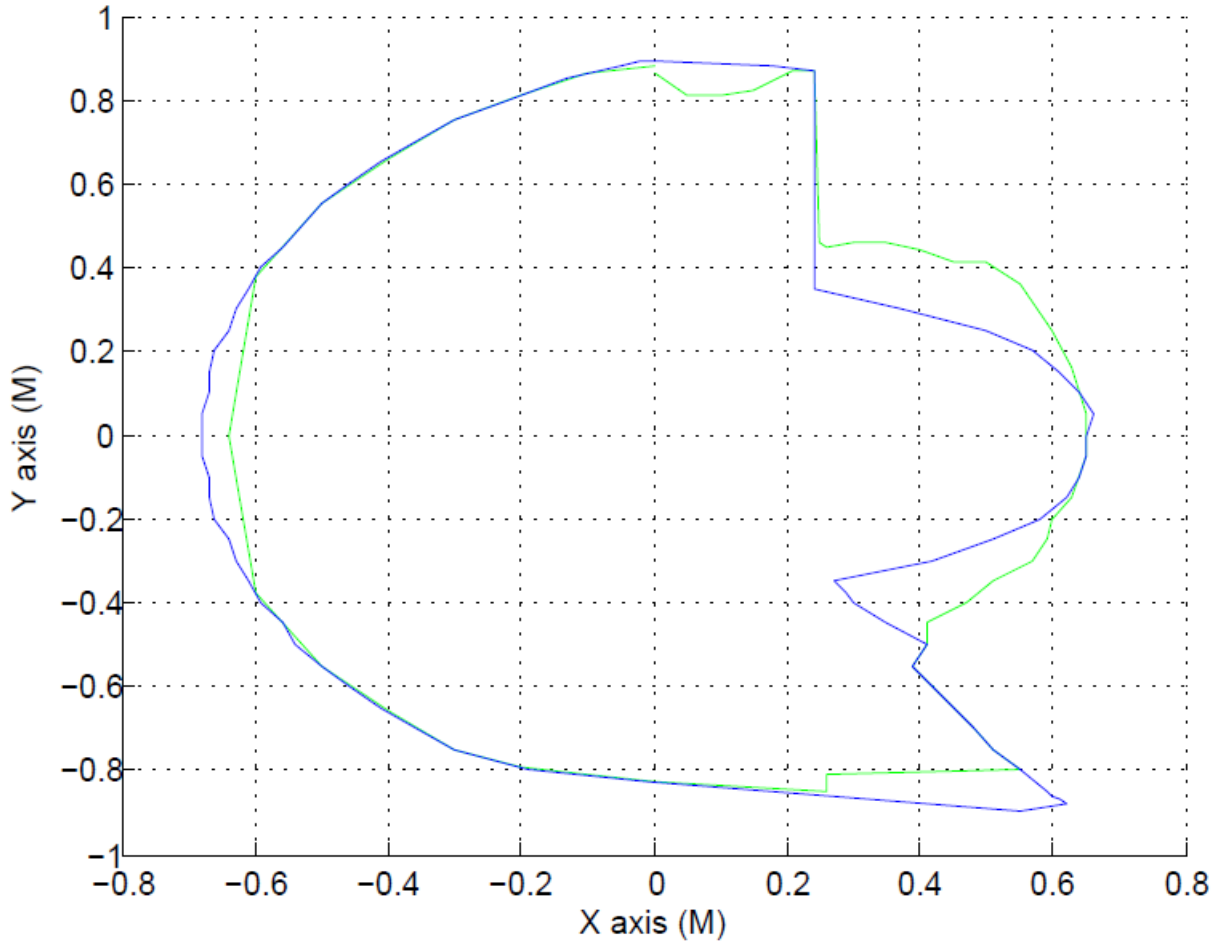


Figure 7.14: Simulated safety inspections vs Experimental safety inspections. Green represents the experimental region and blue represents the simulated region.

Figure 7.13 shows the operational region that the real system was able to cover during the safety inspections experiment. Figure 7.14 shows the comparison between the simulated safety inspections and the experiment. The experimental region in green has a similar shape to the simulated region in blue in Figure 7.14. In the region $x > 0.20$ in Figure 7.14, the experimental region is wider than the simulated region. Simulation results in section 6.6 indicated that configurations in this region have a tendency of driving the end-effector over the roof. Since the kinematic model does not match the real system exactly and the control system does not always give desired results, the end-effector of the real system in this region seemed to be undershooting. The result is a wider region compared to the narrow simulated region. This means the end-effector is able to reach further away without going over the roof.

Simulation results show that the manipulator changes from the formation shown in Figure 6.36 to the formation shown in Figure 6.35 of section 6.6 at $x = 0.27$ and $y = -0.37$. In the experiment this happens at $x = 0.50$ and $y = -0.41$ and this is because the arm covered more space before changing the formation without going over the roof. In the region $x > 0.2$ and $y > -0.80$ the real system covered less space that the simulated system. The reason is that the end-effector of the system was overshooting in this case. The system was unable to cover more space without going over the roof. The same applies for regions $0 < x < 0.20$, $y > 0.80$ and $x < -0.60$. There is no telling whether the end-effector of the real system will overshoot or undershoot. This is a problem that this system faces and can be problematic during operation. This can be overcome by including some sort of feedback and adjusting the joint values of the system according to the feedback received. Another solution would be to use a better control system for the manipulator. These solutions are beyond the scope of this project.

Experimental results presented in this section show the performance of the proposed mine safety inspections system. The system was able to position the end-effector within the area shown in Figure 7.13 without tipping over and without going over the roof. Some parts of the operational region were wider compared to the simulated region due undershooting of the end-effector of the real system. The system was also unable to reach some parts of the simulated area due to

overshooting of the end-effector. For this system to work efficiently, a better control system for the manipulator is needed to avoid undershooting as well as overshooting.

Chapter 8

Conclusions and Recommendations

8.1 Conclusions

This thesis has presented the design of the modeling and path planning components for a robot manipulator mounted on a mobile base for use in the mining safety inspections. Traditional manipulators are used in controlled environments for pick and place, welding and painting applications. Outside of controlled environments these manipulators are teleoperated by a human. This work has argued that a manipulator mounted on a mobile platform can be used for safety inspections in the mining environment. This work has answered the following research questions.

- Which is a suitable way of modeling kinematics of the 5 DOF Packbot510i redundant manipulator and how does this model match the real system?
- Which planning algorithm is suitable for positioning the end-effector of the Packbot manipulator in a desired point in the workspace?
- Which tip-over stability measure is suitable for predicting the point of tip-over of the Packbot system?
- How does the implementation of these algorithms on the Packbot manipulator perform in terms of carrying out the mine safety inspections?

The investigations conducted here for kinematics modeling have shown that a closed-form inverse kinematics method is more suitable than a numerical solution because it readily provides all solutions to the inverse kinematics problem. This allows a solution that better satisfies some criteria to be selected. Numerical methods would typically find a solution and the solution provided is not always acceptable. Numerical methods are computationally expensive due to their iterative nature and therefore are slower than closed-form methods. Closed-form methods include algebraic solutions and geometric solutions, studies have shown that solutions provided by an algebraic technique do not give a clear indication of how to choose a best solution. For this reason an algebraic method was not used and a geometric technique was preferred. Simulation results showed that this geometric technique provided all possible solutions to the inverse kinematics problem of the Packbot510i redundant manipulator. It also made it easy for a solution that better satisfies system stability and minimum total joint angle movement criteria to be selected. It was found that the model developed using this technique suffers from computational errors and as a result the position of the end-effector is sometimes off-set from the desired position.

Experiments were carried out to show how closely the developed model matches the real system. Results obtained indicated that the model is off-set from the real system by a maximum of 5.14%. The causes of this off-set were found to be the computational errors that the model suffers from, gravitational force acting on the system and the poor performance of the control system used in the Packbot software.

A study conducted for path planning algorithms shows that sampling-based algorithms are widely used for industrial grade applications. Combinatorial planning algorithms are rarely implemented due to numerical issues and inefficiency due to combinatorial explosions. Sampling-based RRT planning algorithms were preferred because their incremental nature allows termination of exploration as soon as a solution has been found as opposed to constructing a tree for the entire configuration space. Simulation results showed that RRT* outperforms RRT Ball and RRT in a 2D case in terms of the cost of the path found. RRT* was found to find the shortest path quicker than RRT Ball while RRT found a non-optimal path. Simulation results of the manipulator planning showed that the RRT Ball

is computationally expensive and therefore not suitable for use in the mine safety inspections. In terms of the cost of the path produced for the Packbot manipulator, RRT* and RRT Ball do not offer any significant improvement over RRT. RRT* was found to be slower than RRT when the system stability was introduced but produced better end-effector trajectories than RRT. RRT* was found to be more reliable than RRT and this is because it always strives to produce the shortest path. For these reason RRT* was deemed more suitable to be used in the mine safety inspections.

Experimental results showed that the control system used in the Packbot software is consistent and as a result the end-effector of the real system sometimes does not follow the executed trajectory and ends up slightly off-set from the desired position. It was found that as a result of this, end-effector overshoots as well as undershoots during operation. In other cases the end-effector of the real system was able to follow the executed path with small variations due to vibration of the system while moving. The Packbot robot was designed for teleoperation purposes which explains the issues experienced when using it for autonomous applications.

A study conducted on tip-over stability measures show that Moment Height and Force Angle stability measures are more suitable for use in tip-over prevention of mobile manipulators compared to the Zero Moment Point stability measure. A Force Angle stability measure model for the Packbot510i had been developed and the source code was readily available. This led to the implementation of the Force Angle stability measure in this work. Simulation results showed that the Force Angle stability measure was able to predict point of system tip-over and adjusted the stability index accordingly. Experimental results showed that the Force Angle stability measure is too cautious, i.e, for some configurations with a negative stability index the system was stable. For all stability indices greater than zero the system was always stable which is good. Although the accuracy of the Force Angle stability measure was not proven, this thesis showed that accuracy is of little consequence for the purposes of tip-over prevention.

Simulation results also showed that a complete system formed by integrating the three components is capable of carrying out safety inspections in a mining environment without tipping over the system and violating constraints of the

environment. It was observed that a bigger operational region could be obtained by using a shorter arm because for some end-effector positions the arm has to unfold itself and in the process it will either go over the roof or tip-over the system. Experimental results also showed that the system is capable of carrying out safety inspections in a mining environment. However, it can be used more efficiently with a few modifications. Firstly, since the model does not match the real system exactly, some form of feedback can come in handy. This would be used to adjust the manipulator depending on whether the end-effector is at the desired position or not. Secondly, the inconsistency of the control system used in the Packbot software causes overshooting and undershooting of the end-effector. A better control system is needed or an algorithm that can deal with an inconsistent control system can be used.

Software was designed using C++ using OpenCV2 matrix libraries and a Robot Operating System (ROS) architecture developed at Willow Garage. The software architecture allows for easier message passing and a node-based design using a publish/subscribe framework. Simulation results were conducted on a Dell OptiPlex-760 running Linux Ubuntu 11.04(natty) with 3.2 GB RAM and an Intel(R) Core(TM)2 Duo CPU E8400 3.00 GHz running an electric version of ROS. Experiments were conducted on a Dell XFR running Linux Ubuntu 12.04(precise) with 3.8 GB RAM and an Intel(R) Core(TM) i7-2640M CPU 2.8 GHz running a fuerte version of ROS.

This thesis discussed the practical limitations of the robotic mine safety inspections system, which are primarily based on the type of the robot used, together with constraints of the environment. Experimentation showed that better performance can be achieved by using a robot system with a better control system and a feedback mechanism. This is beyond the scope of this work however, and the system limitations introduced by the control system do not affect the conclusions of this work. This work done in this thesis is being used in the Mining Safety Platform Project at the Mobile Intelligent Autonomous System at the CSIR and will be developed further and improved.

8.2 Recommendations

This work has presented a robotic mine safety inspection system based on the geometric kinematics modeling, RRT* path planning and Force Angle tip-over stability measure algorithms. Experimental results showed that the model does not match the real system exactly therefore a feedback mechanism is recommended to compensate for the model error. A better control system is recommended to be used with the Packbot manipulator. An RRT Based planner that samples one or two joints at a time keeping the rest constant is another way that can be looked at to see if it can offer any improvement over the current RRT based planners in this particular robot manipulator. Results showed that a small operational region was obtained and a bigger operational region could be obtained by using mobile robot with a shorter arm. A recommended manipulator is a 4 DOF manipulator with the first two joints being the same as the first two joints of the Packbot manipulator and third joint being a prismatic joint that slides along the arm increasing the length of the arm. The last joint would be a tilt joint on the end-effector, the same tilt joint as the Packbot manipulator. Due to the gravitational force and forces not taken into account, force control is required to push the end-effector against the roof since the system developed here only positions the end-effector close to the roof. A mechanism of scanning the roof and identifying all the spots that need to be inspected needs to be designed. Finally, obstacle avoidance needs to integrated into this system before it can be used in the mine since there are obstacles in the mine. Since obstacle avoidance is computationally expensive, a bi-directional RRT* is recommended because it finds a path quicker when obstacle avoidance is involved.

Appendix A

.1 Introduction to Robot Manipulators

A robot manipulator is composed of a set of rigid bodies (links) connected together by various joints. Joints can either be very simple, such as revolute or a prismatic, or they can be more complex, such as a ball (spherical) joint. A revolute joint is a 1 DoF rotary joint that rotates by angle θ about the axis of motion (the Z axis for DH convetion). A prismatic joint is a 1 DoF linear displacement d along the joint motion axis

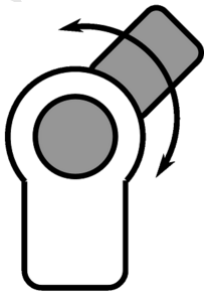


Figure 1: Revolute Joint [73]

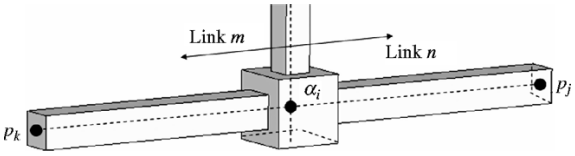


Figure 2: Prismatic Joint [74]

A spherical joint is 3 DoF joint whose configuration is defined by three values

that represent the amount of rotation around the $x-$, $y-$ and $z-$axis. The three values that define a spherical joint's configuration are specified as Euler angles, that are parented in a hierarchy-chain. The analogy is however only valid while all revolute joints keep an orientation distinct from any of the two others. If two joints come close to coincide, a singular situation might appear and the mechanism might lose one DoF. This does not happen with spherical joints that are internally handled to avoid this kind of situation.

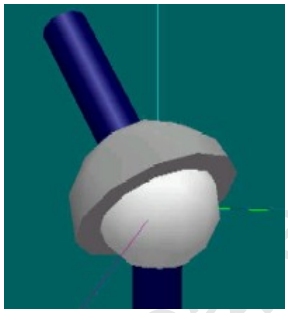


Figure 3: Spherical Joint

There are three types of robot manipulators, serial chain, parallel and hybrid manipulators. A serial chain in general, is an open kinematics chain. The joints must be controlled individuallly. A parallel robot is a closed loop chain and hybrid mechanism is a combination of both closed and open chains. We are dealing with a serial chain manipulator in this project so we will only consider serial chain manipulators.

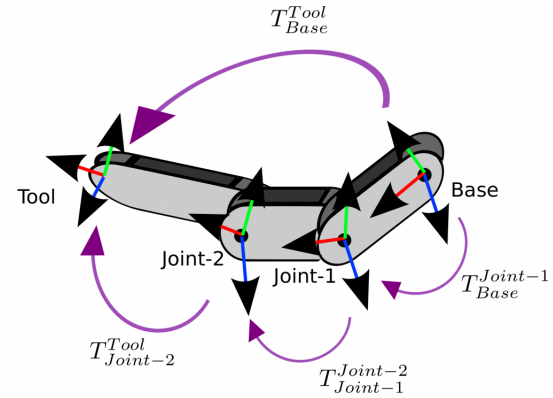


Figure 4: Serial Link Arm [75]

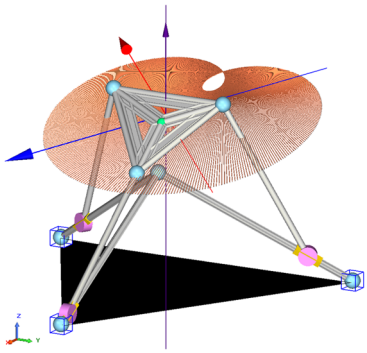


Figure 5: Parallel Link Arm [76]

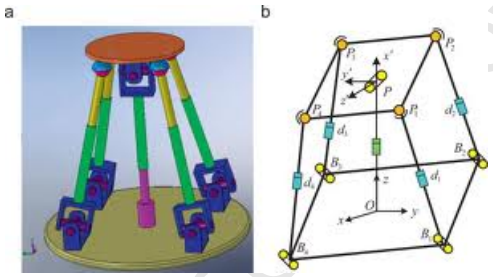


Figure 6: Hybrid Link Arm [77]

Widely used serial link manipulators in the industry are typically composed of links connected together by the joints discussed above. A planar configuration is any combination of 3 rotary and prismatic joints that generate motions of the plane; e.g., $x-z$ translations and planar rotation. RRR is a planar configuration that is constructed by three consecutive revolute joints, whose axes are parallel. RPR is a planar configuration that is constructed by a revolute joint followed by a prismatic joint, which is followed by a revolute joint with all axes parallel to each other.

A cartesian robot manipulator also known as gantry robot consists of three prismatic joints (PPP) translating along the $x-$, $y-$ and $z-$axis as shown below. It is used for pick and place, assembly and packaging applications.

A SCARA robot or Selectively Compliant Articulated Robot Arm is a 4 DoF with 3 revolute joints and one prismatic joint. It is also used for pick and place, assembly and packaging.

A spherical wrist is the most common 6 DoF robot. An example is the PUMA

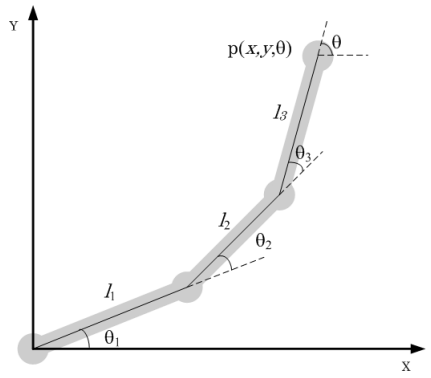


Figure 7: Three link manipulator with revolute joints [78]

Figure 8: The Cartesian Robot Arm [79]

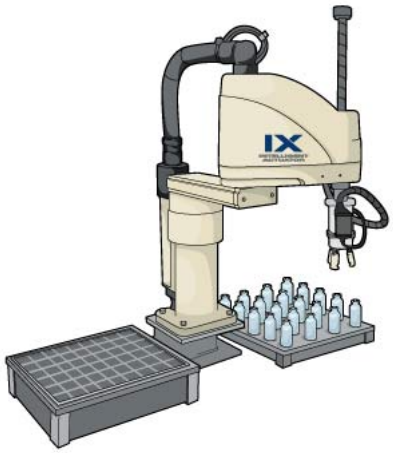


Figure 9: The SCARA Robot Arm [80]

560. The regional structure is a 2 DoF Hooke shoulder joint (formed by combining two revolute joints) followed by a revolute elbow joint. The wrist structure is typically a roll-pitch-roll combination.

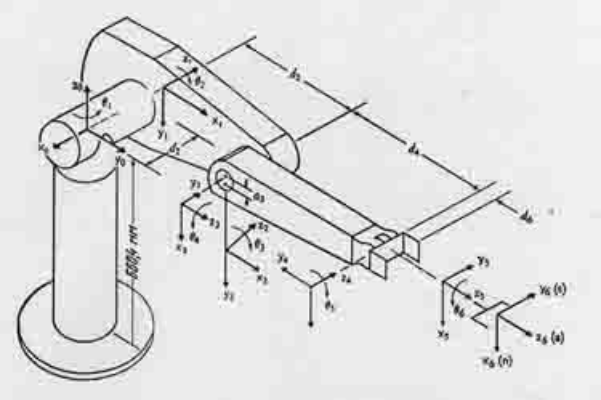


Figure 10: The Puma560 Robot Arm [81]

An anthropomprphic manipulator is designed to resemble a human hand. It is a 6 DoF robotic arm that is designed to achieve what a human arm can achieve and much more. An example is the DLR's Anthropomorphic Hand-Arm-System. The hand consists of 52 drives and more than 100 sensors, and therefore all its tendons feature 19 DoF (the tendons of human's hand equate to 20 DoF).

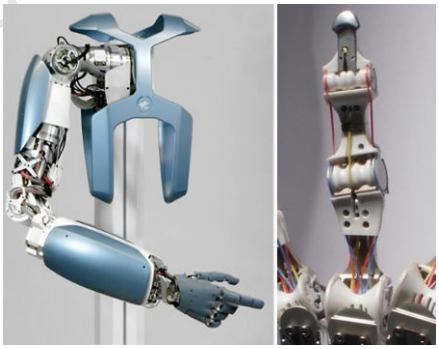


Figure 11: The DLR Anthropomorphic Hand-Arm-System [82]

References

[1] Ngo K.B., Mahony, R., "Passivity-based Control of Robot Manipulators Subject to Constraints." in *Proceedings of the 2005 Australian Conference on Robotics and Automation.* 2005 1

[2] Teleka, R.; Green, J.; Brink, S.; Sheer, J., "Automated Tools to Be Used for Ascertaining Structural Condition in South Africa Hard Rock Mines," *4th Robotics and Mechatronics Conference of South Africa (RobMech 2011),* CSIR International Conference Centre, Pretoria, November 2011 1

[3] Morales, J.; Martinez, J.L.; Mandow, A.; Seron, J.; Garcia-Cerezo, A.J., "Static Tip-Over Stability Analysis for a Robotic Vehicle With a Single-Axle Trailer on Slopes Based on Altered Supporting Polygons," *Mechatronics, IEEE/ASME Transactions on* , vol.18, no.2, to appear. 2, 22

[4] Roan, P.R.; Burmeister, A.; Rahimi, A.; Holz, K.; Hooper, D., "Real-world validation of three tipover algorithms for mobile robots," *Robotics and Automation (ICRA), 2010 IEEE International Conference on* , pp.4431-4436, 3-7 May 2010 3, 23, 113

[5] Rey, D.A.; Papadoupoulos, E.G., "Online automatic tipover prevention for mobile manipulators," *Intelligent Robots and Systems, 1997, Proceedings of the 1997 IEEE/RSJ International Conference on,* vol.3, pp.1273-1278, 7-11 Sep 1997 vii, 3, 22, 67, 68, 69, 70

[6] Dube,C. 2011, "Modeling the manipulator and flipper pose effects on tip over stability of a tracked mobile manipulator," *4th Robotics and Mechatronics*

Conference of South Africa (RobMech 2011), CSIR International Conference Centre, Pretoria, 23-25 November 2011 vii, 5, 24, 67, 71, 73, 113

[7] Kelly, A., "Essential Kinematics for Autonomous Vehicles," Tech. Report CMU-RI-TR-94-14, Robotics Institute, Carnegie Mellon University, May, 1994 9

[8] Carroll, T. 2007, "Robot sensors," *Then and Now,* RobotShop Learning Center, Servo 07.2007. 9

[9] Kallmann, M. 2008, "Analytical Kinematics with body posture control," *Computer Animation and Virtual Worlds,* Vol 19, Issue 2 May 2008, John Wiley and Sons Ltd, Chichester, UK. 9

[10] Craig, J.J., "Introduction to Robotics: Mechanics and Control," Reading, Mass.: Addison-Wesley, 1986. 11

[11] Ito, M.; Kawatsu, K.; Shibata, M., "Maximal admission of joint range of motion based on redundancy resolution for kinematically redundant manipulators," *SICE Annual Conference 2010, Proceedings of,* pp.778-782, 18-21 Aug. 2010 11

[12] Mohamed, H.; Yahya, S.; Moghavvemi, M.; Yang, S.S., "A new inverse kinematics method for three dimensional redundant manipulators," *ICCAS-SICE,* 2009, pp.1557-1562, 18-21 Aug. 2009 11, 13

[13] Xiang Ji; Zhong Congwei; Wei Wei, "A varied weights method for the kinematic control of redundant manipulators with multiple constraints," *Control Conference (CCC), 2011 30th Chinese,* pp.4040-4045, 22-24 July 2011 11

[14] Chiaverini, S., "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators," *Robotics and Automation, IEEE Transactions on,* vol.13, no.3, pp.398-410, Jun 1997 11

[15] Yahya, S.; Moghavvemi, M.; Mohamed, H.A.F., "Improvement of Singularity Avoidance for Three Dimensional Planar Manipulators by Increasing their Degrees of Freedom," *Computer, Consumer and Control (IS3C), 2012 International Symposium on,* pp.568-572, 4-6 June 2012 12

[16] Huang, Q.; Sugano, S.; Tanie, K., "Stability compensation of a mobile manipulator by manipulator motion: feasibility and planning," *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on,* vol.3, pp.1285-1292 vol.3, 7-11 Sep 1997 12

[17] Puiu, D.; Moldoveanu, F., "Real-time collision avoidance for redundant manipulators," *Applied Computational Intelligence and Informatics (SACI), 2011 6th IEEE International Symposium on,* pp.403-408, 19-21 May 2011 12

[18] Moradi H.; Lee S., "Joint Limit Analysis and Elbow Movement Minimization for Redundant Manipulators Using Closed Form Method," *Advances in Intelligent Computing,* Springer Berlin Heidelberg, 2005, vol.3645, pp.423-432 12, 13, 14

[19] Flacco, F.; De Luca, A.; Khatib, O., "Motion control of redundant robots under joint constraints: Saturation in the Null Space," *Robotics and Automation (ICRA), 2012 IEEE International Conference on,* pp.285-292, 14-18 May 2012 12

[20] Spong M.W., Hutchinson S., and Vidyasagar M., "Robot Dynamics and Control," John Wiley & sons Inc., 1989. 12

[21] Ren J.; Zheng Z.; Jiao Z., "Simulation of virtual human running based on inverse kinematics," *Education Technology and Computer (ICETC), 2010 2nd International Conference on,* vol.3, pp.V3-360-V3-363, 22-24 June 2010 13

[22] Wang, L.C.T.; Chen, C.C., "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," *Robotics and Automation, IEEE Transactions on,* vol.7, no.4, pp.489-499, Aug 1991 13

[23] Muller-Cajar, R., Mukundan, R. (2007) Triangualation - "A New Algorithm for Inverse Kinematics," *Hamilton, New Zealand: Image and Vision Computing New Zealand (IVCNZ) 2007 Conference,* 5-7 Dec 2007. Proceedings of Image and Vision Computing New Zealand 2007, PP.181-186. 13

[24] Dash, K.K.; Choudhury, B.B.; Khuntia, A.K.; Biswal, B.B., "A neural network based inverse kinematic problem," *Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE,* pp.471-476, 22-24 Sept. 2011 13

[25] Hu X.; Wang J.; Zhang B., "Motion planning with obstacle avoidance for kinematically redundant manipulators based on two recurrent neural networks," *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on,* pp.137-142, 11-14 Oct. 2009 13

[26] Wu G.; Wang J., "A recurrent neural network for manipulator inverse kinematics computation," *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on,* vol.5, pp.2715-2720 vol.5, 27 Jun-2 Jul 1994 13

[27] Kumar, S.; Behera, L.; McGinnity, T.M., "Kinematic control of a redundant manipulator using inverse-forward adaptive scheme," *Signals and Systems Conference (ISSC 2009), IET Irish,* pp.1-6, 10-11 June 2009 13

[28] Grudic, G.Z.; Lawrence, P.D., "Iterative inverse kinematics with manipulator configuration control," *Robotics and Automation, IEEE Transactions on,* vol.9, no.4, pp.476-483, Aug 1993 13

[29] Lee, C.S.G.; Ziegler, M., "Geometric Approach in Solving Inverse Kinematics of PUMA Robots," *Aerospace and Electronic Systems, IEEE Transactions on,* vol.AES-20, no.6, pp.695-706, Nov. 1984 13

[30] Li Sheng; Wang Yiqing; Chen Qingwei; Hu Weili, "A new Geometrical Method for the Inverse Kinematics of the Hyper-Redundant Manipulators," *Robotics and Biomimetics, 2006. ROBIO '06. IEEE International Conference on,* pp.1356-1359, 17-20 Dec. 2006 13

[31] Singh, G.K.; Claassens, J., "An analytical solution for the inverse kinematics of a redundant 7DoF Manipulator with link offsets," *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on,* pp.2976-2982, 18-22 Oct. 2010 13, 14

[32] Yahya, S.; Moghavvemi, M.; Yang, S.S.; Mohamed, H.A.F., "Motion planning of hyper redundant manipulators based on a new geometrical method," *Industrial Technology, 2009. ICIT 2009. IEEE International Conference on,* pp.1-5, 10-13 Feb. 2009 13

[33] Gan, J.Q.; Eimei, O; Rosales, E.M.; Huosheng, H., "A complete analytical solution to the inverse kinematics of the Pioneer 2 robotic arm," *Robotica, 2005,* vol.23, pp.123 13

[34] De Xu, Carlos A. Acosta Calderon, John Q. Gan etc, "An Analysis of the Inverse Kinematics for a 5-DOF Manipulator," *International Journal of Automation and Computing,* 2005,V2(2): 114-124 13

[35] Manocha, D.; Canny, J.F., "Efficient inverse kinematics for general 6R manipulators," *Robotics and Automation, IEEE Transactions on,* vol.10, no.5, pp.648-657, Oct 1994 13

[36] Gonzalez-Jimenez, L.E.; Carbajal-Espinosa, O.E.; Bayro-Corrochano, E., "Geometric techniques for the kinematic modeling and control of robotic manipulators," *Robotics and Automation (ICRA), 2011 IEEE International Conference on,* pp.5831-5836, 9-13 May 2011 13

[37] Lopez, A.S.; Zapata, R.; Osorio Lama, M.A., "Sampling-Based Motion Planning: A survey," *Computation y Sistemas,* Vol. 12 No. 1, 2008, pp 5-24, ISSN 1405-5546. 14, 16, 17, 18

[38] Latombe, J.C., "Robot Motion Planning," *Kluwer Academic Publishers,* Boston, MA, 1991. 14

[39] Steven M. LaValle, "Motion Planning Part I: the essentials," *IEEE ROBOTICS & AUTOMATION MAGAZINE,* MARCH 2011 vi, 15, 16, 18, 19, 22

[40] Barraquand, J.; Latombe, J.C., "Robot Motion Planning: A Distributed Representation Approach," *International Journal of Robotics Research,* 10(6):628-649, 1991 16

[41] Pierre Bessiere, Juan Manuel Ahuactzin, El-Ghazali Talbi, Emmanuel Mazer., "The Ariadne's clew algorithm: Global planning with local methods," *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)* 16

[42] L.E. Kavraki, P. Svestka, J.C. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation,* 12(4):566-580, 1996. 16, 17

[43] James J Kuffner Jr, Steven M LaValle, "RRT-connect: An efficient approach to single-query path planning," *Proceedings. ICRA'00. IEEE International Conference on Robotics and Automation,* 2000, 995-1001 16, 19

[44] Michael S Branicky, Steven M LaValle, Kari Olson, Libo Yang, "Deterministic vs. probabilistic roadmaps," *IEEE Transactions on Robotics and Automation,* 2002/1. 16

[45] Geraerts, R.; Overmars, M.H., "A comparative Study of Probabilistic Roadmap planners," *In Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02),* 2002, pp. 43-57. 16, 18

[46] Hsu, D.; Latombe, J.-C.; Motwani, R., "Path planning in expansive configuration spaces," *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on,* vol.3, pp.2719-2726 vol.3, 20-25 Apr 1997 16

[47] Kindel, R.; Hsu, D.; Latombe, J.-C.; Rock, S., "Kinodynamic motion planning amidst moving obstacles," *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on,* vol.1, no., pp.537-543 vol.1, 2000 16

[48] Sanchez, G.; Latombe, J.C., "On Delaying Collision Checking in PRM Planning – Application to Multi-Robot Coordination," *Int. J. of Robotics Research,* 21(1):5-26, January 2002. 17

[49] Sanchez, G.; Latombe, J.C., "A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking," *Int. Symposium on Robotics Research (ISRR'01),* Lorne, Victoria, Australia, November 2001. Published in

Robotics Research: The Tenth Int. Symp., R.A. Jarvis and A. Zelinsky (eds.), Springer Tracts in Advanced Robotics, Springer, pp. 403-417, 2003. 17

[50] Zhao, M.; Ansari, N.; Hou, E.S.H., "Mobile manipulator path planning by a genetic algorithm," *Journal of Robotic Systems,* Vol. 11 NO. 1, Wiley Subscription Services, Inc., A Wiley Company, 1994, pp 143–153. 17

[51] Yang, J.; Dymond, P.; Jenkin, M., "Practicality-Based Probabilistic Roadmaps Method," *Computer and Robot Vision (CRV), 2011 Canadian Conference on,* pp.102-108, 25-27 May 2011 18

[52] Karaman, S; Frazzoli, E., "Sampling-Based Algorithms for Optimal Motion Planning," *The International Journal of Robotics Research,* June 2011, vol. 30, no. 7, pp.846-894 18, 20, 22, 46

[53] James J. Kuffner Jr. and Steven M., "RRT-Connect: An efficient approach to sigle query path planning," *Proc. IEEE Int. Conf. Robot. Autom. (ICRA),* 2000, pp 995–1001.

[54] Steven M. Lavalle and James J. Kuffner and Jr., "Rapidly-Exploring Random Trees: Progress and Prospects," *Algorithmic and Computational Robotics: New Directions,* 2000, pp.293-308. 19, 46

[55] Lindemann, S.R.; LaValle, S.M., "Incrementally reducing dispersion by increasing Voronoi bias in RRTs," *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on,* vol.4, pp. 3251- 3257 Vol.4, April 26-May 1, 2004 19, 20

[56] Zhang, L.; Manocha, D., "An efficient retraction-based RRT planner," *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on,* pp.3743-3750, 19-23 May 2008 20

[57] Ferguson, D.; Stentz, A., "Anytime RRTs," *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on,*pp.5369-5375, 9-15 Oct. 2006 20

[58] Akgun, Baris; Stilman, Mike, "Sampling heuristics for optimal motion planning in high dimensions," *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on,* pp.2640-2645, 25-30 Sept. 2011 20

[59] Krammer, L.; Granzer, W.; Kastner, W., "A new approach for robot motion planning using rapidly-exploring Randomized Trees," *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on,* pp.263-268, 26-29 July 2011 21

[60] Lacevic, B.; Rocco, P.; Strandberg, M., "Safe motion planning for articulated robots using RRTs," *Information, Communication and Automation Technologies (ICAT), 2011 XXIII International Symposium on,* pp.1-7, 27-29 Oct. 2011 21, 22

[61] McGhee, R.b.; Frank, A.A., "On the stability properties of quadruped creeping gaits," *Mathematical Biosciences,* vol.3, pp.331-351, 1968. 22

[62] Huang, Q.; Sugano, S.; Kato, I., "Stability control for a mobile manipulator using a potential method," *Intelligent Robots and Systems '94. 'Advanced Robotic Systems and the Real World', IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on,* vol.2, pp.839-846, 12-16 Sep 1994 23

[63] Ali, S.; Moosavian, A.; Alipour, K., "Stability Evaluation of Mobile Robotic Systems using Moment-Height Measure," *Robotics, Automation and Mechatronics, 2006 IEEE Conference on,* pp.1-6, Dec. 2006 23

[64] Papadopoulos, E.G.; Rey, D.A., "A new measure of tipover stability margin for mobile manipulators," *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on,* vol.4, pp.3111-3116, 22-28 Apr 1996 23, 24

[65] Rey, D.A.; Papadoupoulos, E.G., "Online automatic tipover prevention for mobile manipulators," *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on,* vol.3, pp.1273-1278, 7-11 Sep 1997 24

[66] Beck, C.; Miro, J.V.; Dissanayake, G., "Trajectory optimisation for increased stability of mobile robots operating in uneven terrains," *Control and Automation, 2009. ICCA 2009. IEEE International Conference on,* pp.1913-1919, 9-11 Dec. 2009 24

[67] Miro, J.V.; Dumonteil, G.; Beck, C.; Dissanayake, G., "A kyno-dynamic metric to plan stable paths over uneven terrain," *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on,* pp.294-299, 18-22 Oct. 2010 25

[68] Hatano, M.; Obara, H., "Stability evaluation for mobile manipulators using criteria based on reaction," *SICE 2003 Annual Conference,* vol.2, pp.2050-2055 Vol.2, 4-6 Aug. 2003 25

[69] Mann, M.P.; Shiller, Z., "Dynamic Stability of a Rocker Bogie Vehicle: Longitudinal Motion," *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on,* pp. 861- 866, 18-22 April 2005 26

[70] Morales, J.; Martinez, J.L.; Mandow, A.; Seron, J.; Garcia-Cerezo, A.; Pequeo-Boter, A., "Center of gravity estimation and control for a field mobile robot with a heavy manipulator," *Mechatronics, 2009. ICM 2009. IEEE International Conference on,* pp.1-6, 14-17 April 2009 26

[71] Liu, Y.; Liu, G., "Interaction Analysis and Online Tip-Over Avoidance for a Reconfigurable Tracked Mobile Modular Manipulator Negotiating Slopes," *Mechatronics, IEEE/ASME Transactions on,* vol.15, no.4, pp.623-635, Aug. 2010 26

[72] Makondo,N; Claassens, J; Tlale, N; Braae, M. 2012, "A Geometric Technique for the Kinematic Modeling of a 5 DOF Redundant Manipulator," *5th Robotics and Mechatronics Conference of South Africa (RobMech 2012),* CSIR International Conference Centre, Pretoria, 26-27 November 2012. 44

[73] Bohemia Interactive Simulations, "createJointExt-RevoluteJoint"s[Online], Available at:http://resources.bisimulations.com/wiki/createJointExt_-_RevoluteJoint[Accessed 15 March 2013]. 144

[74] Francisco Rubio, Francisco Valero, Joseph Sunyer, Vicente Mata, (2009), "Direct step-by-step method for industrial robot path planning," *Industrial Robot: An International Journal,* Vol. 36 Iss: 6, pp.594 - 607. 144

[75] Elysium Labs, "Joint state parameterization and forward kinematics"[Online], Avaliable at:http://www.elysium-labs.com/robotics-corner/learn-robotics/introduction-to-robotics/joint-state-parameterization-and-forward-kinematics/[Accessed 15 March 2013] 145

[76] CompMech Research Group, "Parallel Robots with Mixed Freedoms"[Online], Available at:http://www.ehu.es/compmech/parallel-robots-with-mixed-freedoms/ [Accessed 15 March 2013] 146

[77] Zhang, D. Gao, Z., "Forward kinematics, performance analysis, and multi-objective optimization of a bio-inspired parallel manipulator," *Robotics and Computer-Integrated Manufacturing,* Volume 28, issue 4, August 2012, PP:484-492. 146

[78] Adelhard Beni Rehiara, "Kinematics of Adept Three Robot Arm"[Online], Robot Arms, 2011, Available at:http://www.intechopen.com/books/export/citation/BibTex/robot-arms/kinematics-of-adeptthree-robot-arm[Accessed 15 March 2013] 147

[79] Direct Industry website, "The Virtual Industrial Exhibition"[Online], Available at:http://www.directindustry.com/prod/montech/cartesian-robots-5419-898635.html[Accessed 15 March 2013] 147

[80] IAI Quality and Innovation, "IX SCARA Series Robots"[Online], Available at:http://www.intelligentactuator.com/ix-scara-series-robots/[Accessed 15 March 2013] 147

[81] Vyacheslavovych, A., "Research of dynamic properties of the industrial robot with using diffeometrial a regulator and a nonlinear feedback"[Online], Available at:http://masters.donntu.edu.ua/2007/kita/animitsa/diss/indexe.htm[Accessed 15 March 2013] 148

[82] I New Idea Homepage, "DLR's Anthropomorphic Hand-Arm System"[Online], Available at:http://www.inewidea.com/2011/02/05/36607.html[Accessed 15 March 2013] 148